Master's thesis

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

David André Bjerkan Mikkelsen

A study of control-bounded estimation filter architectures

Master's thesis in Electronic Systems Design
Supervisor: Trond Ytterdal
Co-supervisor: Fredrik Esp Feyling
June 2022

NTNU
Kunnskap for en bedre verden

David André Bjerkan Mikkelsen

A study of control-bounded estimation filter architectures

Master's thesis in Electronic Systems Design
Supervisor: Trond Ytterdal
Co-supervisor: Fredrik Esp Feyling
June 2022

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

PREFACE

I would like to thank my supervisor, professor Trond Ytterdal, for introducing me to such an interesting field. The guidance provided throughout the projects these past 18 months has been invaluable.

Also, I would like to thank Fredrik Esp Feyling and Dr. Hampus Malmberg. The additional support has been helpful, and the interest in my results have been motivating. And naturally, I would not have this exciting opportunity without their pioneering work on the subject.

ABSTRACT

Analog-to-digital converters (ADCs) are important in many applications. Recently, a new type of converter has been emerging called control-bounded converters. It offers a different perspective to the ADC process. This framework is less restrictive, making new and interesting implementations possible. There is a lack of data on the associated digital filter as this conversion concept is very recent. This thesis will try to rectify that by exploring digital filter implementations for a single-input system.

Three implementations will be presented in this thesis. A batch implementation using recursive calculations for both the lookahead and lookback. A finite impulse response (FIR) implementation which directly calculates the result from lookup tables. And a hybrid between these, using recursive calculations for the lookback and lookup tables for the lookahead. Additionally, a variation of each implementation will be presented. An implementation that downsamples in two stages will try to reduce the area of the implementations with recursions. And an approach using duplicate circuits will attempt to further lower the power consumption of the FIR implementation.

The area and power consumption of these implementations will be explored. The thesis will attempt to establish trends over different parameters of the control-bounded system. Parameters such as filter depth, downsampling rate, system order, and clock frequency.

In general, this thesis will show the FIR implementation having the most favorable scaling. It will achieve 313µW dynamic power consumption with 60dB signal-noise ratio and 40MS/s. The thesis will show that the FIR implementation is best suited for this application in most cases. There will also be shown a few cases where the hybrid implementation may be better.

SAMMENDRAG

Analog-til-digital omformere (ADC) er viktige i mange applikasjoner. Nylig har en ny type omformer vokst frem, kalt kontrollbegrenset (control-bounded) omforming. Den tilbyr et annerledes perspektiv av ADC prosessen. Dette rammeverket er mindre begrensende, noe som gjør nye og interessante implementasjoner mulige. Det mangler informasjon om det tilhørende digitale filteret ettersom denne metoden for omforming er så ny. Denne oppgaven vil prøve å lindre denne mangelen ved å utforske digitale filter implementasjoner for et enkeltinngangs system.

Tre implementasjoner vil bli presentert i denne oppgaven. En batch implementasjon som bruker rekursive kalkulasjoner både for framoverblikk og bakoverblikk. En avgrenset impuls respons (FIR) implementasjon som beregner resultatet direkte, basert på oppslagstabeller. Og en hybrid mellom disse, hvor rekursive kalkulasjoner bruker i bakoverblikket og oppslagstabeller brukes til foroverblikket. I tillegg vil en variasjon av hver implementasjon bli presentert. Én implementasjon hvor nedsampling skjer i to steg som vil prøve å redusere arealet til implementasjonene med rekursjoner. Og en fremgangsmåte som bruker kopier av kretsen i et forsøk på å senke kraftbruken til FIR implementasjonen ytterligere.

Arealet og kraftbruken til disse implementasjonene vil bli utforsket. Oppgaven vil prøve å etablere trender over forskjellige parametre i det kontrollbegrensede systemet. Parametre som: filter dypde, nedsamplingsrate, systemorden, og klokkefrekvens.

Generelt vil denne oppgaven vise at FIR implementasjonen har den mest gunstige veksten. Den vil oppnå en dynamisk kraftbruk på 313µW med 60dB signal-støy rate og 40MS/s. Oppgaven vil vise at FIR implementasjonen passer best til denne applikasjonen i de fleste tilfeller. Det vil også bli vist et par tilfeller hvor hybrid implementasjonen kan være bedre.

Table of Contents

List of Figures

List of Tables

Acronyms

ADC analog-to-digital converter.

DAC digital-to-analog converter.

DSR downsampling rate.

FIR finite impulse response.

HLS high-level synthesis.

LSB least significant bit.

LUT lookup table.

MASH multi-stage noise shaping.

MIMO multiple-in-multiple-out.

MSB most significant bit.

OSR oversampling rate.

PnR place and route.

PSD power spectral density.

RAM random access memory.

RTL register transfer level.

SISO single-in-single-out.

SNR signal-noise ratio.

1 Introduction

These days we are surrounded by digital systems, but all of them must interact with an analog world. Converters between the analog and digital domains will remain important. There are many applications with different needs, and many approaches used to achieve them.

One application is ultrasound imaging which requires fairly fast conversion, typically in the range of 10MS/s to 40MS/s. The converter's precision will affect the quality of images. At the same time, since this is an imaging application it will have many converters placed close together. This means an increase in power consumed by a converter will be a massive increase for the chip as a whole. So a low power converter is desired.

A new approach for analog to digital conversion has been emerging, called control-bounded conversion. Which inverts the approach used by conventional ADCs. This gives more freedom for analog design, creating a lot of room for innovation. While the analog part of the converter may see a lot of interesting new solutions it will still need a digital filter.

There is a lack of data on the power consumption of this digital filter since the approach is so new. Therefore this thesis aims to establish a foundation to better know what to expect from the filter. While ultrasound imaging is the application being worked against, a general foundation of data for this converter is desired.

1.1 Related Work

Dr. Malmberg's dissertation on control-bounded conversion is the primary source for the theoretical foundation of this approach [1]. It describes an overview of this method, as well as how it was derived. And also how it compares to conventional sample-based ADCs. Some implementations of the analog system are described, and a description of a control-bounded digital-to-analog converter (DAC) is included. But, most important for this thesis is the description of the estimation filter's task and function.

Fredrik Feyling has been working on realizing the analog system and digital control on a transistor level [2]. That thesis describes design challenges encountered when designing an implementation of the analog portion of a leapfrog system. It also expands on the theory by analyzing the several non-idealities not previously accounted for. The analog system parameters used in this thesis were chosen by Feyling.

The author of this thesis has been working with the digital estimation filter during two other projects prior to this work. First in a project using high-level synthesis (HLS) to attempt implementing the batch estimator [3]. Which gave the first draft of the batch architecture, and the experience highlighted some issues with the HLS flow.

Then in [4], the three basic estimator architectures were implemented in a 28nm CMOS technology using SystemVerilog, where the top-level designs largely remain unchanged. That project established some trends in power and area. However, these power estimations were inaccurate, and they only showed trends across filter depth and downsampling rate (DSR). The results indicated that the finite impulse response (FIR) implementation had the lowest total power consumption in all configurations tested.

1.2 Scope

The goal of this thesis is to provide trends in power consumption and area for the digital estimation filter of a control-bounded ADC. This shows which parameters are most important when optimizing the estimator. Thus, the trends hopefully provide insight that can be used to form a basis for design decisions. Only implementations for a single-in-single-out (SISO) ADC are evaluated.

This thesis builds upon the previous work done by improving the accuracy of the estimates. Which makes the results more reliable. It also fixes some errors in the implementations which were limiting the signal-noise ratio (SNR) and giving incorrect simulation results after synthesis.

Another way this thesis builds upon previous work is by providing trends across a wider variety of parameters. Trends in power consumption and area are established across filter depth, DSR, control-vector size, frequency, and technology node.

Three architectures of the digital estimation filter have already been implemented: A FIR architecture, a batch architecture, and a hybrid architecture. These have been described in an earlier work [4], but are included so this thesis serve as a comprehensive document. The figures have been revised to be more accurate to their current implementation, and the descriptions are rewritten. This work also implements a version of each architecture with an extra clock domain, with the goal of reducing their size or power consumption. In the batch- and hybrid architectures the new clock domain is used to reduce the size of the recursive calculations by downsampling in two stages. In the FIR implementation the new clock domain is used to further reduce the clock frequency in the calculation circuit. This is done by adding copies of the circuit and pushing their results out sequentially, this is called the staggered architecture.

These implementations are synthesized primarily in a 22nm CMOS FDSOI technology which is commercially available, but also uses a 28nm technology to see the differences between these technology nodes. The numbers presented in this thesis have been estimated from synthesized netlists using Synopsys Primetime, which is a power signoff tool. These estimations include parasitics by running the synthesis tool in topographical mode, so it performs an approximate place and route (PnR) and can extract capacitances and resistances of wires.

The estimation filter implementations were written in SystemVerilog, and are parameterized to be flexible. They can be found on GitHub [5].

Memory modules are outside the scope of this thesis. Therefore the random access memory (RAM) modules used in the batch implementations are only simulated, but then excluded from the rest of the design flow. This does mean that estimates for the batch implementations are slightly lower than they really would be.

1.3 Specifications

Table 1.1: Specifications

Parameter	22nm	28nm
Supply voltage	0.59V	0.8V
Analog system	4. order leapfrog	
Clock frequency	240MHz	
Sample rate	40MS/s	
SNR	> 60dB	
Sample bit-width	12 bits	
Power Consumption	< 100µW	

The target application of this work is ultra-sound imaging. This means the signal bandwidth is 5MHz. It would be a noisy environment so the sample rate out from the ADC is higher than the Nyquist rate, using 40MS/s. Although, the work attempts to establish general trends by deviating from these parameters.

1.4 Outline

The thesis starts by summarizing theory that is useful to understand the rest of the sections. It gives a brief overview of how the control-bounded converter functions, and an explanation of the estimation filter. There is a brief description of delta-sigma converters to compare with the control-bounded. As well as other theory that is useful to be aware of when interpreting the presented results.

Then, in section 3 a high-level description of all estimation filter implementations are included. It should provide a thorough explanation of how these architectures work.

Before the results are presented, there is a summary of the toolchain used to obtain them. Then the results are presented and discussed.

Finally, it concludes that the FIR implementation usually will be best for systems with a single input. But a hybrid implementation gave the best result for the specifications used. There is also a summary of the trends seen in each architecture.

The appendix contains several additional plots that were deemed redundant or less important. A snippet of python code used to generate the analog system, and a failed implementation from the previous work [4] are also included.

2 Theory

2.1 Number representations

This thesis will be dealing with fractional (or real) numbers. To represent these in binary there are two common choices: fixed-point, and floating-point representation. These formats have been explained in many books on digital arithmetics, for example in [6] and [7].

The fixed-point format is widely used in hardware implementations. In this format, an integer is mapped such that some or all the least significant bits (LSBs) represent the fractional part of the number. That is, it uses a fixed position for the radix point which is chosen during the design phase.

The main advantage of the floating-point representation is its dynamic range. That is the difference between the largest and smallest numbers that can be represented in this format. This is achieved by letting the radix point move, or "float", according to the most significant digits of the number being represented. This does come at the cost of complexity, as extra logic is needed to move the radix point. Also, since some of the bits are used to set the radix position it is said [6] to have lower precision.

$$y = (-1)^{sign} * significand * 2^{exponent-bias} \tag{2.1}$$

The fields of a floating-point number are explained in [6]: Sign, exponent, and mantissa. Where the sign is a single bit noting whether the number is positive or negative. The mantissa is the fractional part of the significand, which includes an implicit bit in the most significant position. The significand is typically normalized to always have a magnitude between 1 and 2. The exponent tells what position the radix point currently is in. It is stored as a biased, unsigned number where the bias of an n-bit exponent is $2^{n-1} - 1$. When the exponent is zero it means the number is in the subnormal range. This means the significand's implicit bit is 0, leading to the significand being less than 1. Equation 2.1 shows how these fields relate to the number being represented.

Comparing these two representations, floating-point arithmetic is more complex than fixed-point. A floating-point addition requires control logic and barrel shifters to align the radix point of the two numbers and handle special conditions. This generally leads to higher area cost in hardware implementations, which can be prohibitive. As stated in [7] the size of a fixed-point multiplier is viewed to be N times larger than an adder, where N is the word size. Meanwhile, the area of an adder and multiplier in floating-point are nearly the same. This means the ratio of adder and multiplier operations will impact the size difference between using these two representations. The dynamic range needed in the application would also be important for the choice of representation. As a fixed-point representation may need significantly wider data paths to cover the same range a floating-point number would.

Earlier work [4] on control-bounded converters has concluded that the dynamic range needed does not warrant the use of floating-point representation. Therefore this thesis will primarily use fixed-point representation.

2.1.1 Rounding

Often there will be a need to reduce the number of mantissa bits in the data, this happens in multiplications and conversions between formats. When truncating a fixed-point number the default behavior is to always round down (toward $-\infty$). This introduces an offset error on the produced result, due to an asymmetry in the rounding of positive and negative numbers, which is discussed in [8]. This offset will appear in power spectral density (PSD) plots as a low-frequency noise signal.

There are different rounding schemes used to mitigate this effect, the simplest of which is "round half up". In that rounding scheme, a fixed number is added before the data is truncated, leading to data being rounded to the nearest number. For example, when converting from the format

Q4.4 to Q4.0 (4 fraction bits rounded to an integer) the added number would be 0.5. If the data's fraction is larger than 0.5 the carry bit will spill into the new range. As stated in [8] there is still one situation where the numbers are not symmetrical, but it removes most offset introduced by rounding.

Rounding in floating-point does not encounter the same issue since it uses a sign-and-magnitude representation instead of a complementary one. This means truncating the floating-point mantissa will not have problems with symmetry, but it will still benefit from using the "round half up" scheme.

2.2 Oversampling and downsampling

As explained by the theory [9]: Oversampling occurs when the signals of interest are bandlimited to f_0 while the sample rate is at f_s, where $f_s > 2f_0$ and $2f_0$ is the Nyquist rate. That is, the signal is sampled at a much higher frequency than is required to capture its content. The oversampling rate (OSR) is defined as equation 2.2.

$$OSR = \frac{f_s}{2f_0} \tag{2.2}$$

Downsampling is also explained in [9], it typically occurs in discrete-time signal processing and is the process of reducing the rate of data. This thesis refers to the DSR, which is the ratio of the incoming sample rate divided by the outgoing rate. It is performed simply by removing samples, it will keep one out of every DSR incoming samples. The incoming signal has to be band-limited to half the frequency of the outgoing rate. Otherwise, aliasing will move noise to the signal band. If the incoming data is not band-limited a digital low-pass filter will be necessary before downsampling.

2.3 CMOS power considerations

A collection of low power techniques and considerations can be found in [10]. It says that the power consumption of a CMOS circuit can be placed in three categories: switching, leakage (or static), and short-circuit. This thesis will collectively refer to the switching and short-circuit power as dynamic power.

The short-circuit power is caused by both transistors in a complementary pair being open during a state transition. This is affected by the transition time of a gate input. That is, a network with high capacitance will cause increased leakage power in the cells connected to it.

Leakage power is the power consumption of a gate in a steady state. Which is caused by various effects such as sub-threshold currents, and diode leakage. The leakage power in a circuit generally increases exponentially with both lower threshold voltage and higher temperature. This is discussed extensively in [10]. Naturally, the leakage power also increases with the number of gates in a circuit.

$$P_{sw} = C * V_{DD}^2 * A * f \tag{2.3}$$

Switching power is caused by the charging and discharging of capacitances in a network. This can be described with equation 2.3, from [10]. In this equation C is the capacitance, V_{DD} is the supply voltage, A is the activity factor (the switching probability), and f is the frequency. This shows that the supply voltage has a quadratic relation to power consumption, while the other factors are linear.

Many methods exist to reduce the power consumption of a circuit, targeting the various mechanisms outlined above. Lowering the supply voltage is a way to reduce all three categories of power consumption, but a side-effect of this is the circuit becoming slower. That's where multiple supply voltages are useful; Allowing the use of a high voltage in sections requiring high performance, and a low voltage for low-performance sections. Another way to reduce power is by reducing the size of gates. This will reduce its parasitic capacitance, but again, reduces its speed. There are also different ways to implement the same logic function, where one may be faster than another, but less power efficient. This is not a comprehensive list, additional methods are discussed in [10].

Slack is a measure often used in digital circuits. It is the difference between the required arrival time and estimated arrival time for a data path. The synthesis tool will calculate the slack for every network of the design and can use it to optimize the power consumption. A positive slack means the design can tolerate becoming slower. The synthesis tool will use this to automatically adjust the gate sizes and implementation of logic.

2.4 Delta-Sigma conversion

A well-known ADC strategy is delta-sigma conversion, which is an oversampling converter. Figure 2.1 shows what a continuous-time delta-sigma ADC may look like with a first-order modulator. The modulator consists of a differential adder (delta) between the input signal and feedback, a low-pass filter or integrator (sigma), and a quantization and sampling step. The quantization noise power is considered constant, so oversampling will spread it over a wider band leaving less noise power in the signal band. Additionally, the feedback loop means the quantization noise will be high-pass filtered by the integrator. This improves the quantization noise in a first-order system by 9dB per doubling of OSR, as stated in [9]. There are higher-order systems that further improve this noise filtering, such as a multi-stage noise shaping (MASH) delta-sigma architecture, interested readers are referred to [9].

The result of this process is that the bit-stream s[k] can be used to generate a sample with more bits of precision than the quantizer has. The final step of the conversion is the decimation filter, which calculates a weighted average value for this bit-stream, and downsamples the data.

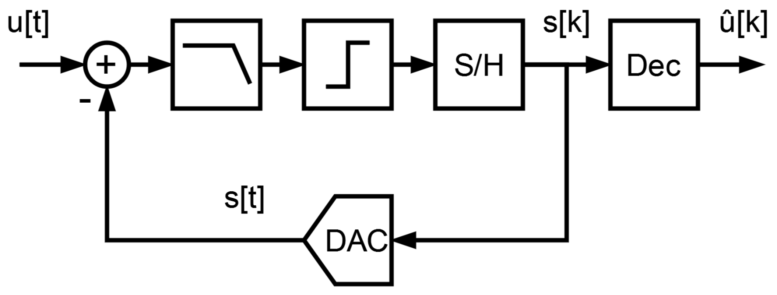


Figure 2.1: Continuous-time delta sigma ADC with a first order modulator

2.5 Control-bounded conversion

The control-bounded converter uses a different approach for the conversion task compared to conventional sampling converters. A sampling converter generally seeks to acquire an approximate value s[k] for the input u[t], which is an inherently lossy process due to the quantization of the value. Whereas a control-bounded converter seeks the input u[t] based on an observed state s[k]. That is, the problem being solved is inverted. The dissertation of Dr. Hampus Malmberg [1] is the primary source of this theoretical framework.

A control-bounded converter primarily consists of three parts, as explained in [1]: An analog system, a digital control, and a digital estimation filter. The analog system is also known as the preconditioning filter, its task is amplifying the wanted signal characteristics while suppressing unwanted characteristics. The digital control stabilizes the analog system by counteracting its growth via the feedback loop. The estimation filter calculates the estimated sample based on the digital control states.

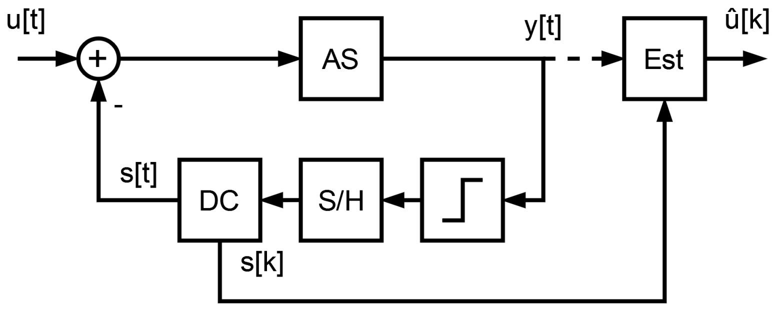


Figure 2.2: Control-bounded ADC overview

Figure 2.2 shows an overview of what a control-bounded filter could look like. It is drawn to resemble the delta-sigma converter in figure 2.1. Where the digital control (DC) in this figure could be implemented as a DAC, and the analog system (AS) could be an integrator or low-pass filter.

One key difference between the two is the perception of signal s[k]. In the delta-sigma converter, this is considered a quantized and sampled version of the measured signal. Whereas the control-bounded converter only views this as the control signal which stabilized the analog system, and is only indirectly related to the measured signal. As stated in [1], this leads to a fundamentally different approach for the digital post-filtering step.

Another point is that the signals in figure 2.2 are usually vectors. Which means it is closer to a MASH architecture than the first order delta-sigma pictured in figure 2.1. In fact it is stated in [1] that control-bounded ADCs can be considered a generalization of the MASH delta-sigma concept, with a less restrictive design space.

From [1] the analog systems (AS) can be implemented as a chain of integrators or low-pass filters. Other architectures are also proposed, such as the chain of oscillators architecture. There is also the leapfrog architecture where the integrator chain has feedback across nodes, and the Hadamard architecture which seeks to reduce the impact of component mismatch. That is to say: there is a lot of possibility in the analog design space for control-bounded converters.

This thesis may refer to the number of analog states and digital states, which is the width of the vectors y[t] and s[k] respectively. The number of analog states may also be called the system order N, and reflects the number of elements in the analog system's chain. The number of digital states M is the total number of control signals, it will typically be the system order multiplied by the quantization depth. This thesis has only evaluated systems using 1-bit quantizers, meaning M = N.

2.5.1 Estimation filter

As stated, the function of the estimation filter is to calculate the value of the sample estimate. It does so using the control vectors to estimate the trajectory of each state in the analog system, and uses these states to find what the measured signal is. In an ideal control-bounded conversion, the algorithm will converge at an analog value of the measured signal as the considered time window extends to infinity [1]. This means the estimate is not affected by quantization. Instead, it will be limited by other factors such as the precision used in the calculations, the depth (time window) of the filter, or the conversion error.

Some figures of the control-bounded converter have an imaginary connection of the analog state observation y[t] to the estimation filter. This signifies that the filter uses the control vectors to reconstruct the internal state trajectories of the analog system. It uses these states to calculate what the measured signal u[t] has to be.

However, the estimation filter does not have perfect knowledge of the analog states, which leads to a conversion error. This error can be quantified as a linear mapping of the remainder in the analog states after a control period. The quality of the results depend on the size of the estimated state trajectories compared to this error. That is, as the state trajectories become larger than this error the results become more accurate, which is explained in [1].

There is a lot of complicated math behind the function of the estimation filter. However, knowledge of how the algorithm is derived is not necessary as a set of linear formulas has been presented in [1]. The standard formulas use matrix operations, which would lead to an exponential increase in complexity as the system order rises. Therefore the parallel version of the formulas have been used. With this set of equations, the calculation of each state trajectory is independent and the complexity rises linearly.

$$\overrightarrow{\tilde{m}}_{k,n} = \overrightarrow{\tilde{\lambda}}_n \overrightarrow{\tilde{m}}_{k-1,n} + \overrightarrow{\tilde{f}}_n(s[k-1]) \tag{2.4}$$

$$\overleftarrow{\tilde{m}}_{k,n} = \overleftarrow{\tilde{\lambda}}_n \overleftarrow{\tilde{m}}_{k+1,n} + \overleftarrow{\tilde{f}}_n(s[k]) \tag{2.5}$$

$$\hat{u}_l[k] = \sum_{n=1}^{N} (\overrightarrow{\tilde{w}}_{n,l} \overrightarrow{\tilde{m}}_{k,n} + \overleftarrow{\tilde{w}}_{n,l} \overleftarrow{\tilde{m}}_{k,n}) \tag{2.6}$$

These equations describe the recursive calculations forming the basis of this conversion. Equation 2.4 is the forward recursion, which starts calculating at the oldest control vector and proceeds forward in time toward the newest vector. The backward recursion in equation 2.5 does the opposite, starting at the newest vector and going backward in time. How these recursive results relate to the final sample is seen in equation 2.6.

All quantities in these equations are scalar, with complex values. Luckily, the imaginary components cancel each other, resulting in an estimate that is purely real. The index k refers to the discrete-time of each factor. Index l discerns which input is being calculated, but this thesis has only evaluated single input systems. Index n refers to the analog state, as a set of recursions are needed for each state trajectory being calculated. The functions $\overleftarrow{\tilde{f}}_n(*)$ and $\overrightarrow{\tilde{f}}_n(*)$ map the contribution of a control vector toward each state trajectory. The function is simply a constant value related to each bit of the control vector.

Appendix A shows how all constants were generated using a python package.

2.5.2 FIR estimation

$$\hat{u}[k] = \sum_{l1=0}^{L1} \overleftarrow{h}_{l1} s[k + l1] + \sum_{l2=1}^{L2} \overrightarrow{h}_{l2} s[k - l2] \tag{2.7}$$

The estimator can be implemented as a FIR filter, which can be described by equation 2.7. In this description L1 is the lookahead length, or the window length toward the future control vectors. L2 is the lookback length, describing the window length for past vectors.

This FIR estimator uses the coefficients $\overleftarrow{h}$ and $\overrightarrow{h}$ to calculate the estimate. They are vector values that, similar to the $\tilde{f}$ functions of the recursive equations, map a constant contribution for each bit of the control vector. However, this is the total contribution toward the final estimate. Meaning it neither needs complex arithmetic nor separate calculations for each state trajectory, it calculates the result directly from a window of control vectors.

2.5.3 Hybrid estimation

$$\hat{u}[k] = \sum_{l1=0}^{L1} \overleftarrow{h}_{l1} s[k + l1] + \sum_{n=1}^{N} \overrightarrow{w}_{n,l} \overrightarrow{m}_{k,n} \tag{2.8}$$

Another alternative for the implementation of the estimation filter is a hybrid between the FIR and recursive approaches. Equation 2.8 shows this approach. Implementing the forward recursions in hardware is simple, but the backward recursions are not trivially implemented since they go backward in time. The hybrid approach avoids this issue by using a FIR lookahead instead.

2.5.4 Downsampling

$$\overrightarrow{\tilde{m}}_{k,n} = \overrightarrow{\tilde{\lambda}}_{n}^{R} \overrightarrow{\tilde{m}}_{k-1,n} + \sum_{x=0}^{R-1} \overrightarrow{\tilde{\lambda}}_{n}^{x} \overrightarrow{\tilde{f}}_{n}(s[Rk - x - 1]) \tag{2.9}$$

$$\overleftarrow{\tilde{m}}_{k,n} = \overleftarrow{\tilde{\lambda}}_{n}^{R} \overleftarrow{\tilde{m}}_{k+1,n} + \sum_{x=0}^{R-1} \overleftarrow{\tilde{\lambda}}_{n}^{x} \overleftarrow{\tilde{f}}_{n}(s[Rk + x]) \tag{2.10}$$

Equations 2.9 and 2.10 show the downsampled version of the recursive calculations. The DSR in these equations is noted by the factor R, which is an integer value.

These equations show that the estimation needs every control vector to form an accurate estimate. This means no information can be removed before the estimate is calculated. Instead, a lumped calculation is performed where several vector contributions are added in parallel. Doing this makes it possible to reduce the frequency of the calculation so the power consumption can be reduced.

$$\hat{u}[k] = \sum_{l1=0}^{K1} \overleftarrow{h}_{l1} s[Rk + l1] + \sum_{l2=1}^{K2} \overrightarrow{h}_{l2} s[Rk - l2] \tag{2.11}$$

The formula for a downsampled FIR implementation is shown in equation 2.11. In this case, the calculation needs no change. The only difference is that the window of control vectors is moved DSR positions each time.

Using the downsampling methods outlined above won't remove any information from the time domain before the estimation step is performed. This means no aliasing should occur in the estimation step. However, the samples produced may show aliasing if the reconstructed signal is not band-limited to half the sample rate.

The specifications used in this thesis have an output sample rate 4 times higher than the Nyquist rate, and 8 times higher than the analog system's filter frequency. This gives the analog system a margin for suppressing out-of-band components. The thesis has assumed that no digital anti-alias filtering would be necessary.

3 Implementations of the digital estimator

3.1 Common modules

3.1.1 Clock divider

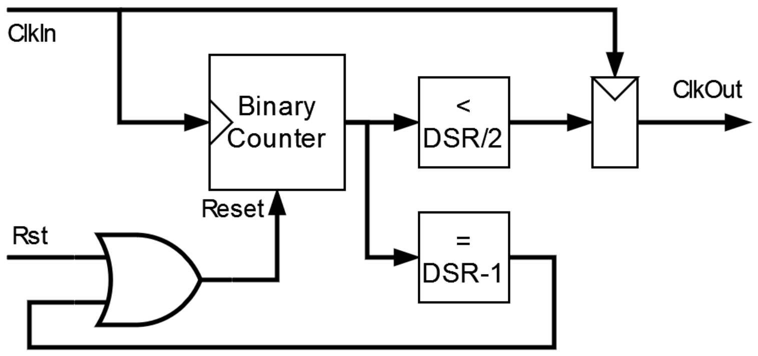


Figure 3.1: Clock divider implementation

The clock divider is used to generate the slower clocks in each architecture. It divides the frequency of a clock by an integer value. Which is implemented using a counter so it can divide the frequency by an arbitrary integer. The output is high when the counter value is less than DSR/2, which means the positive clock edge occurs when the counter is reset to 0. This reset signal is set when the counter reaches DSR-1, and it is synchronous to ensure that it switches at the correct time.

The clock pulses out from this module are not symmetrical when odd an DSR is used, therefore only the positive edge from it is used. A register on the output synchronizes the new clock to the input clock, this makes the timings between clock domains more predictable and avoids glitches from the combinational logic.

3.1.2 Valid counter

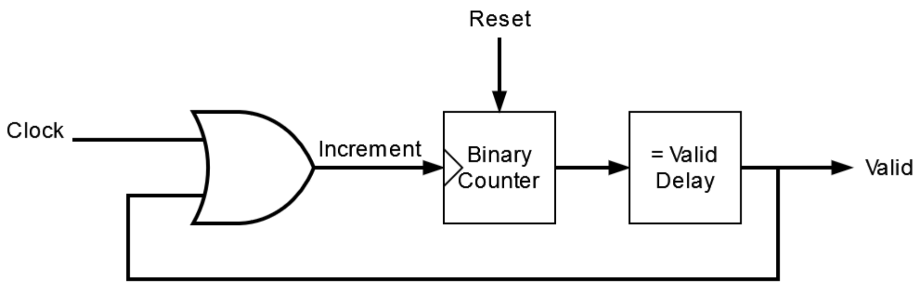


Figure 3.2: Valid counter implementation, figure from [4]

This counter keeps track of the time since a reset occurred, so it can decide when the output samples are valid. The number of clock cycles required before results are valid depends on filter length or batch size. Therefore "Valid Delay" is decided during compilation of the code. The counter uses the same clock signal as the output of each architecture to minimize its activity. When the valid signal is set high this module will gate off its own clock, stopping all dynamic power.

3.1.3 LUT

Table 3.1: The table contents of a 3-bit LUT, table from [4]

input	output
0b111	$f_2 + f_1 + f_0$
0b110	$f_2 + f_1 - f_0$
...	...
0b001	$-f_2 - f_1 + f_0$
0b000	$-f_2 - f_1 - f_0$

Lookup tables (LUTs) are used to convert the control vectors into numbers used to calculate the sample values. As seen in table 3.1 each bit in the control vector has a constant number associated with it. That bit being high or low determines whether its constant is added or subtracted from the total contribution. These tables are dynamically generated during compilation. The synthesis tool will implement them as logic since all its values are constant.

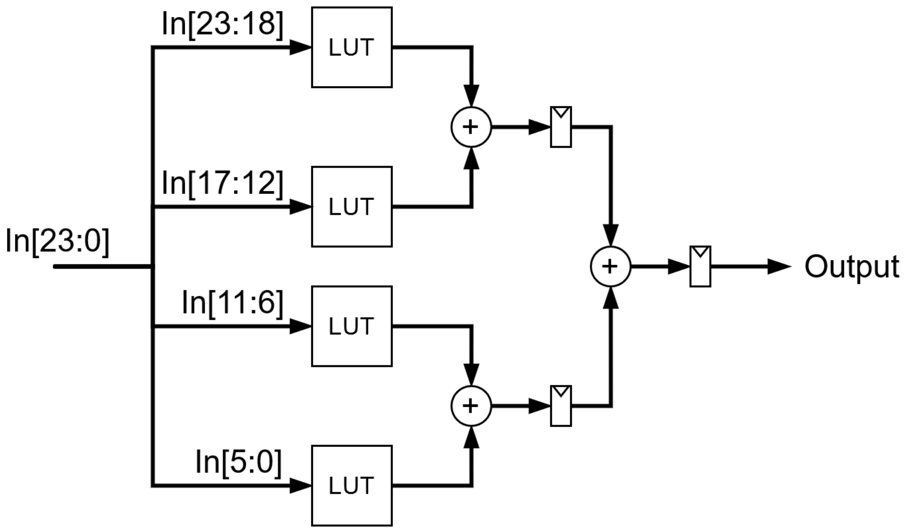


Figure 3.3: A 24-bit LUT split into four 6-bit LUTs, figure from [4]

A large LUT will be split into several small LUTs and summed to reduce the area. Since the size of a single LUT increases exponentially with input bits. The optimal size depends on the numerical format used due to the different complexity of their adders. Floating-point LUTs have a higher optimal point than fixed-point does. The fixed-point implementations use 4-bit LUTs, while the floating-point implementations use 6-bits.

3.1.4 Output Formatter

The output of all estimation filters implemented is a 12-bit fixed-point number. However, most of these implementations use a different format internally to achieve the desired SNR. There are also some implementations using floating-point numbers, which need to be converted.

When fixed-point numbers are used converting the output is fairly simple. The formatter module rounds to the closest value by adding a bias value, then truncates the value. This is also known as a ”round half up” scheme. Section 2.1.1 explains why this is necessary.

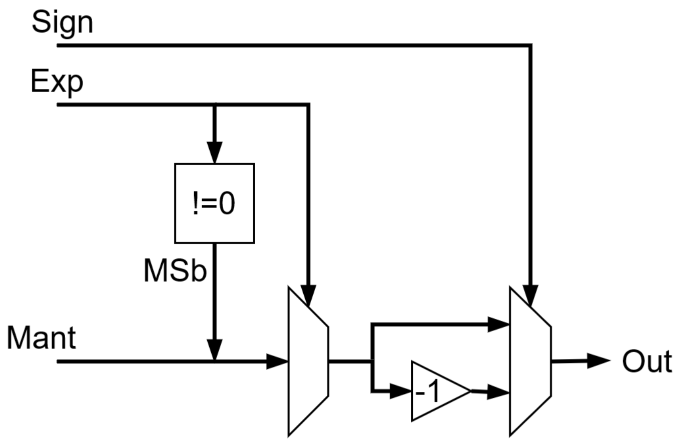


Figure 3.4: A hardware implementation of floating-point to fixed-point conversion

Changing from floating-point to fixed-point numbers is more complicated, as seen in figure 3.4. This converter checks if the number is sub-normal by checking if the exponent is zero, this determines whether the implied bit of the mantissa should be one or zero. Then a barrel shifter moves the radix point of the number to a constant position. The number is then truncated using the "round half up" scheme, doing so while the number is unsigned avoids any symmetry issues. Finally, the sign of the floating-point determines if the number is positive or negative.

The implementations use an unsigned value for its final output, while they use signed values internally. Converting from a signed to an unsigned fixed-point number is the very last step in each implementation. Since the signed numbers are represented with 2-complement this conversion can be done simply by inverting the most significant bit (MSB).

3.2 FIR estimator

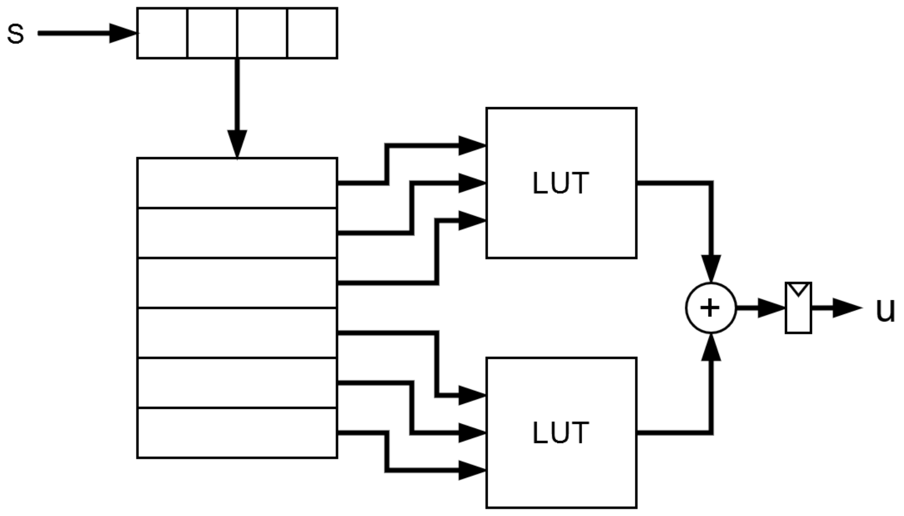


Figure 3.5: FIR implementation, figure from [4]

The FIR architecture is the simplest of these implementations and is depicted in figure 3.5. It consists of a shift-register, and two LUTs. Control vectors are pushed into the shift-register which forms the time window being evaluated. One LUT is for the lookahead part of the calculation, the other for the lookback, and these partial results are summed to form the estimated value.

3.2.1 Downsampling

By downsampling, the design will produce estimates at a reduced rate. This architecture does so by collecting several control vectors in a small shift-register, then pushing them to the main shift-register. The only part of this design running at the main clock frequency is the input shift-register. Doing this reduces the activity in the main shift-register and its connected LUTs, which reduces the power consumption of this filter.

3.2.2 Staggered

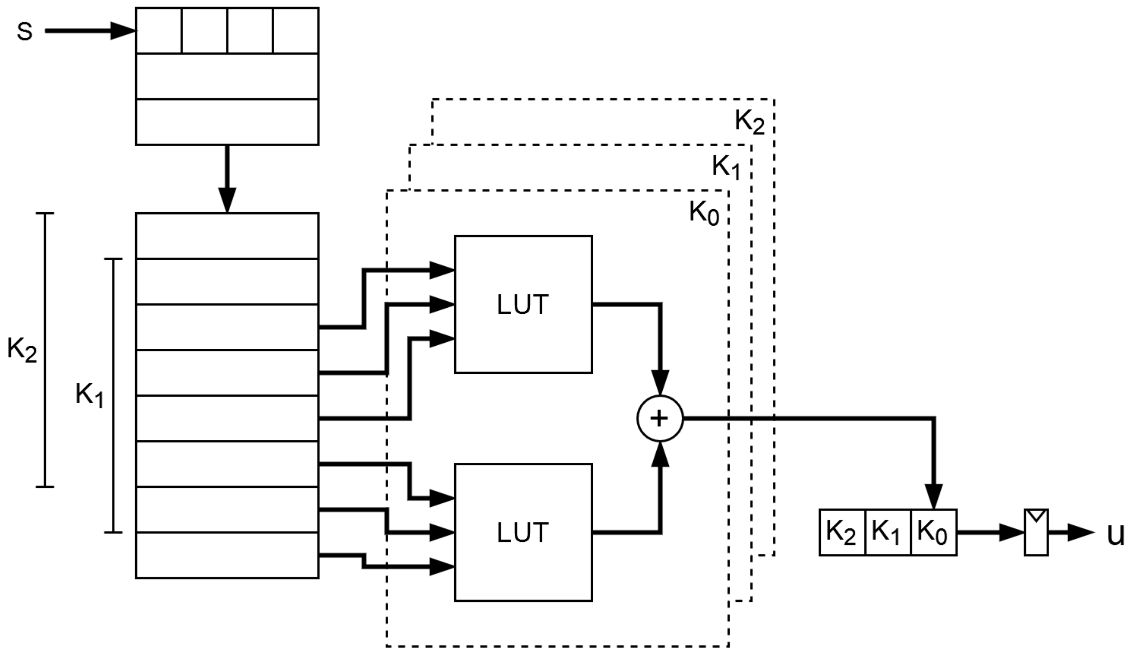


Figure 3.6: FIR staggered implementation

Since frequency is closely related to power consumption it may be worth reducing it further. To calculate samples at a lower rate than samples are output the filter would need duplicates of the calculation circuit, which is shown in figure 3.6. This will naturally increase the size of the estimator. But most of the shift-register is shared between these duplicates. The greatest benefit of this would be seen in a multi-voltage system where the supply voltage could be reduced in the duplicate circuits.

To minimize the switching activity this architecture collects control vectors for each duplicate in the input register, then pushes all of them to the main shift-register. The results from each circuit are shifted out at the final sample rate. This means the clock frequency of the calculation circuits and main shift-register is the output frequency divided by the number of duplicates.

3.3 Batch estimator

This architecture uses recursive calculations both for the forward and backward mean. It is a direct implementation of the equations from section 2.5.1. To use recursive calculations for the backward mean it must handle the control vectors and samples in batches.

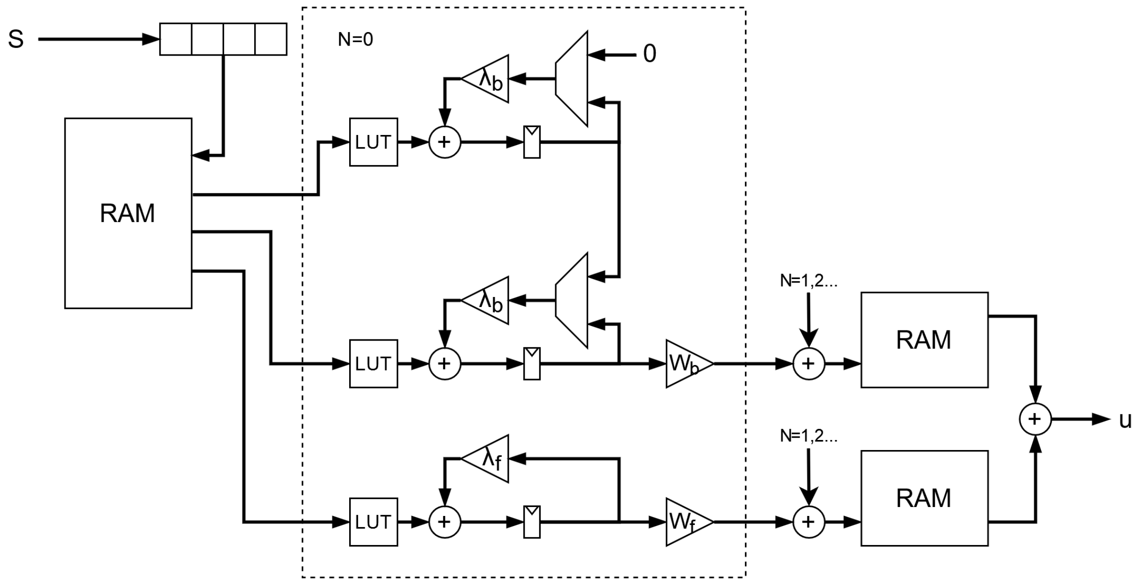


Figure 3.7: Batch implementation with system order N

Figure 3.7 shows this implementation. The control vector S is placed in a RAM module, where it will be read three times during the estimation process. Each analog state trajectory (noted by N) has three recursive calculations: The forward recursion is at the bottom and handles control vectors forward in time, that is from oldest to newest. In the middle, the backward recursion is shown going backward in time, from newest to oldest. The backward recursion needs a starting point to avoid abrupt edges when batches are changed. This is the purpose of the topmost recursion, the lookahead. It performs the same calculation as the backward recursion, only on a future batch. Each analog state trajectory has a weighted contribution toward the result, as decided by the factors W_b and W_f. All these contributions are summed before being saved to the RAM modules for the partial results.

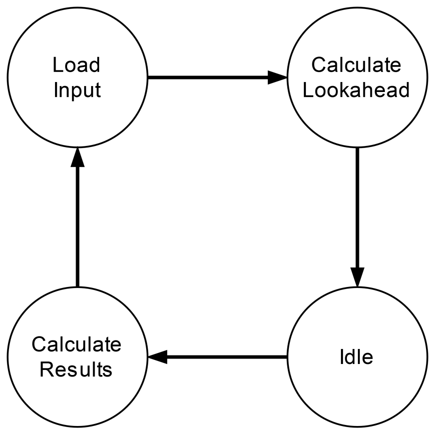


Figure 3.8: The batch cycle, figure from [4]

The recursive calculations use complex numbers, up until factor W_b or W_f. After that point, the imaginary part is discarded and only real arithmetic follows. This can be done because the imaginary parts will become zero when all partial results are summed.

Each batch of control vectors in the first RAM module goes through the four cycles shown in figure 3.8. These cycles occur in parallel to provide constant throughput of results. Therefore, this RAM module has room for four batches and has three read ports and one write port. When control vectors are read they will first be in the "load input" step, where they are saved in the RAM. During the next step, they will be used to calculate the lookahead value. The following step is "idle", here the calculation step uses the result from the lookahead to calculate the next batch. Finally, the calculation step will calculate the forward and backward recursions for this set of vectors.

This batch logic is implemented by interleaving the batches in the memory. Where the two LSBs of the address indicate the cycle. There are two counters keeping track of the batch progress, one counting up and the other counting down. The one counting up is used to assign the MSBs of the address for the input loading and forward recursion. The other counter is used for the address of the lookahead step and backward recursion. Each memory port has a 2-bit counter to determine which batch is currently at this step of the cycle. This all results in the access pattern shown in figure 3.9. The left side of the figure shows which cycle each memory location belongs to, and the right side illustrates the two first clock cycles k of a batch.

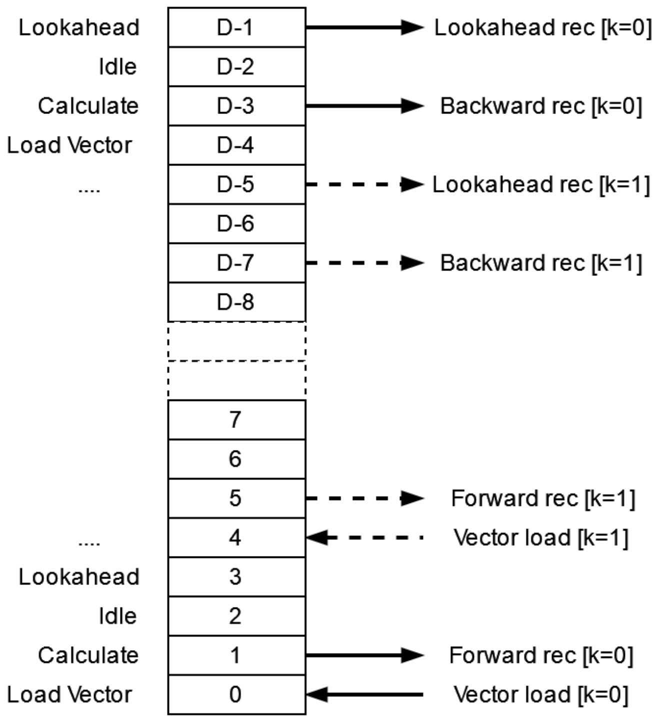
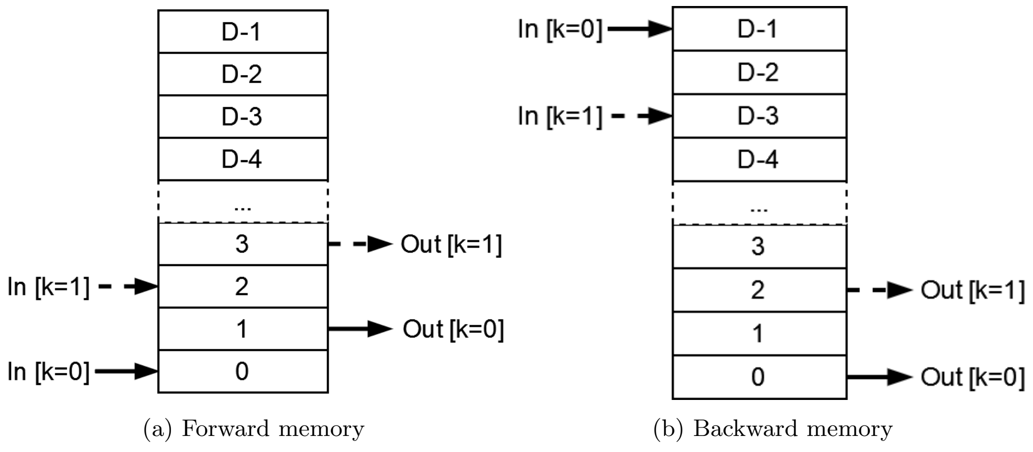


Figure 3.9: Access pattern in a control-vector memory of size D, figure from [4]

When the system transitions between two batches the lookahead recursion will transfer its final result to the backward recursion. At the same time, the lookahead recursion's memory is discarded to reset it without downtime. This transition can not occur when the cycle changes, because the results from the LUTs are delayed. Therefore the transfer must be delayed such that the last vector contribution of that batch has been processed by the recursion.

When the system enters the reset state its valid counter will indicate when the results are valid, just as the FIR implementation does. Additionally, the calculation recursions will remain cleared until new control vectors reach them to prevent random data from being saved. This should let the recursions converge on an accurate estimate faster when a reset occurs.



(a) Forward memory

(b) Backward memory

Figure 3.10: Access patterns in estimate memory, figure from [4]

The memories for partial estimates from the forward and backward recursions are separate, and they each have one read and one write port. The partial estimates are also interleaved in the memory, this time using 1 LSB. Since the backward recursion works backward in time the order of its estimates must be reversed to form the final sample. This access pattern can be seen in figure 3.10b. The forward estimates use the access pattern shown in figure 3.10a.

The input and output addresses are swapped at every batch transition. These addresses are based on the same counters used by the control vector RAM. However, they have to be delayed to account for the delay in the calculation path to ensure the batches transition seamlessly. The forward results also must be delayed by one additional clock cycle to correspond with the formulas.

Table 3.2: Required size of memories

Memory	Size
Control-Vector	$4 * batchsize * controlvectorsize$
Backward estimate	$2 * batchsize * (outputsize + 1)/DSR$
Forward estimate	$2 * batchsize * (outputsize + 1)/DSR$

Table 3.2 shows the minimum size of each memory. The implementations used the same size for the forward and backward estimate memories, as this made the control logic simpler. However, the forward memory can technically be reduced to half this size by writing directly after the result is read.

To minimize the size of the partial estimate memories the results were formatted before being saved. One extra bit should be saved as there is one addition remaining before the result is final. Otherwise, this reduction may introduce unwanted noise.

3.3.1 Downsampling

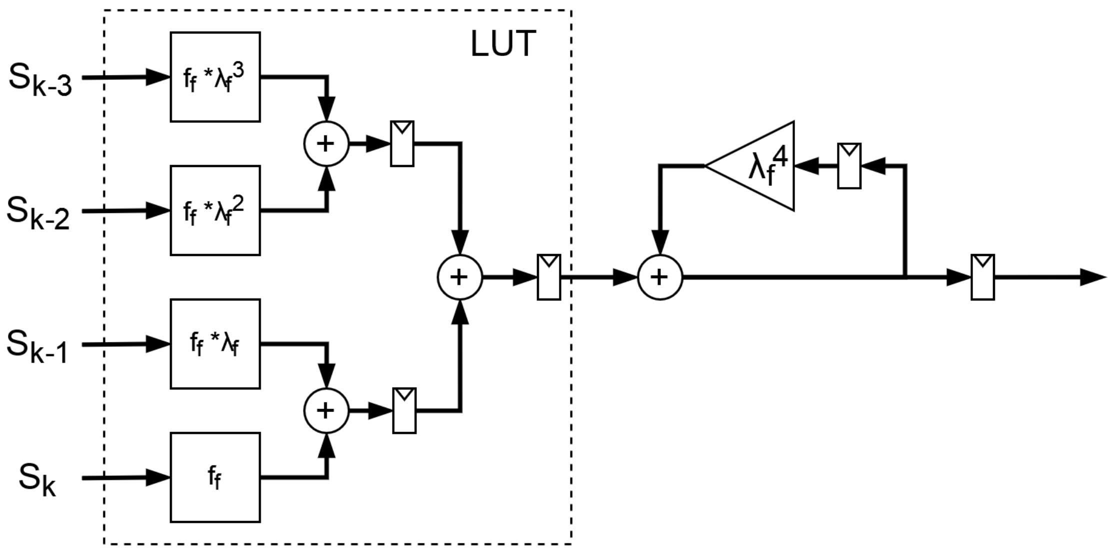


Figure 3.11: The forward recursion with a DSR=4, figure from [4]

Control-vector S is loaded in a shift-register, this combines them in a larger vector. This allows the rest of the circuit to run at a lower frequency. However, to form an accurate estimate the recursive calculations need the information from every vector. Therefore they are handled in parallel, which increases the size of these LUTs. To ensure the vector's contribution is correct the loop-factor λ is multiplied with a power depending on the time offset of each vector. The recursive loop then uses the loop-factor to the power of DSR. One example of this can be seen in figure 3.11.

To ensure no batches overlap or have gaps the batch size is rounded up to the nearest multiple of DSR. The memory required for the partial estimates naturally shrinks as DSR increases. Since the number of estimates from a batch is divided by DSR.

Another consequence of downsampling is the data ports in the RAM for control vectors becoming wider. This allows the RAM to operate with the reduced clock frequency. Its total size is not changed, which leads to the address space becoming smaller.

3.3.2 Two-stage downsampling

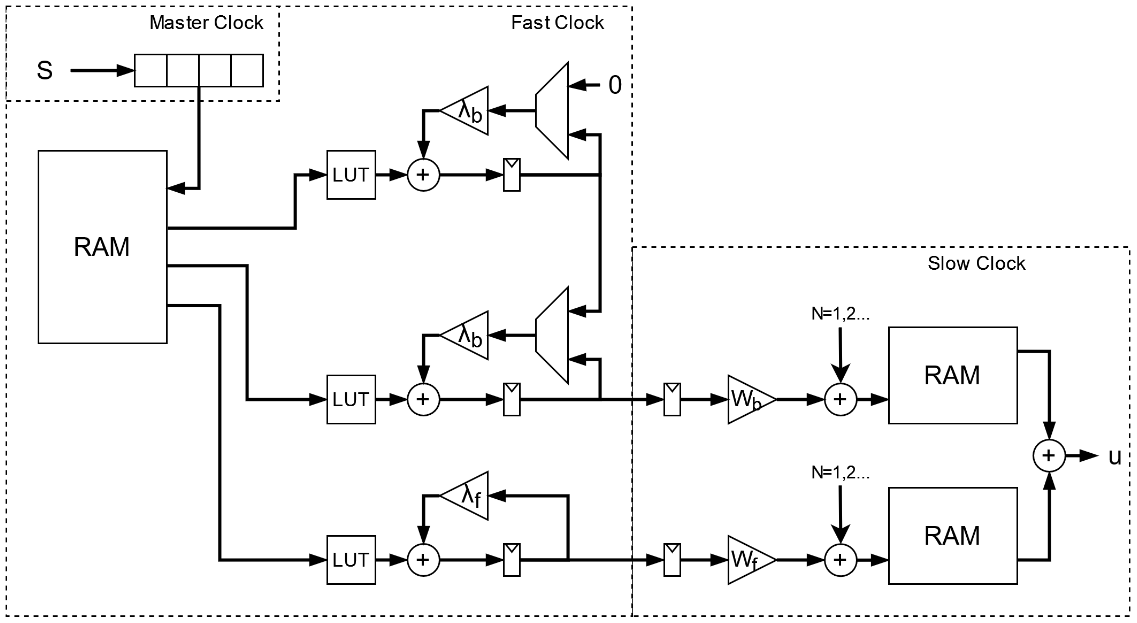


Figure 3.12: Batch two-stage implementation with system order N

By adding another clock domain the size of the LUTs can be reduced. Since the LUT size for the recursions scale with DSR, using a lower DSR will reduce their size. Figure 3.12 shows the clock domains in this architecture. The transition from the master clock to a slower "fast clock" is identical to the regular architecture. Then another downsampling occurs after the recursions to achieve the correct sample rate for the output. The frequency of the recursions will increase compared to the regular architecture, but it can prevent excessive growth.

The fast frequency is DSR_1 times slower than the master frequency, and the slow frequency is DSR_2 times slower than the fast frequency. Both of these transitions must be integer relations. The total DSR is the product of these two DSRs. This means some DSR values will have more possible combinations than others.

3.4 Hybrid estimator

The hybrid architecture combines the lookahead from the FIR architecture and the forward recursion from the batch architecture. This removes the need for batch logic while providing an infinite lookback. This leads to this architecture usually having a higher SNR than the pure FIR architecture, and its shift-register is half as big. It does however need a recursive calculation for each analog state in the system, leading to the size scaling with DSR.

Just like the batch implementation, this implementation uses complex arithmetic in the recursion until the state trajectories are multiplied with factor W_f. It also determines when the recursion should start calculating by using the valid counter.

The delay of the lookahead and lookback is not the same. Typically the lookahead delay is longer since the LUT is much larger. To compensate for this the lookback can read control vectors from an earlier point in the shift-register. That way the corresponding partial results can be added without inserting registers to delay them.

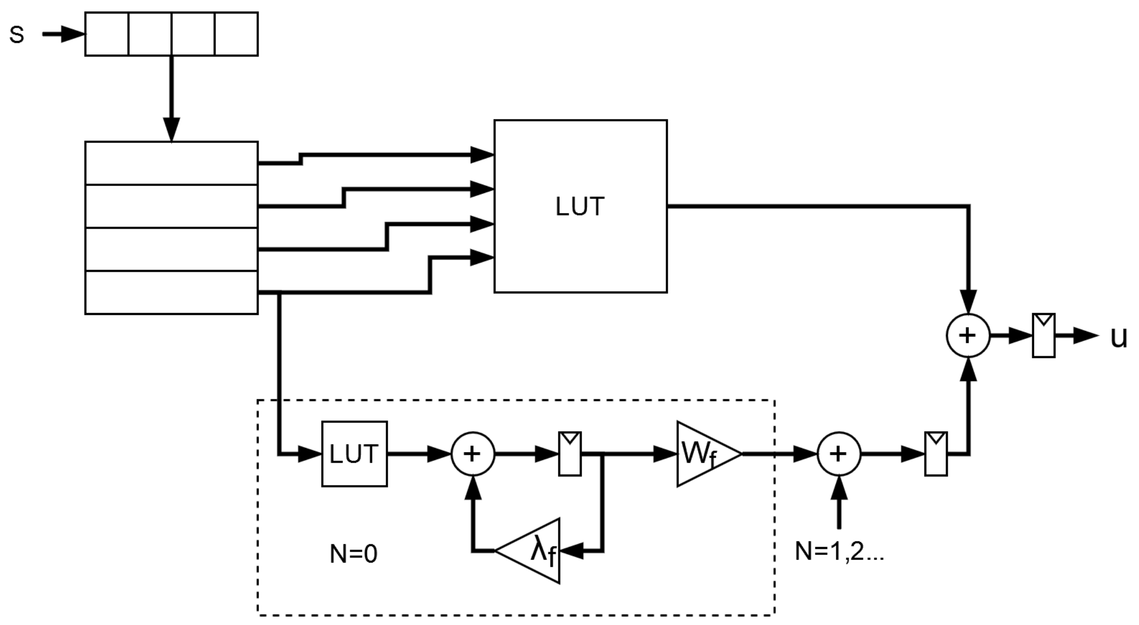


Figure 3.13: Hybrid implementation with system order N

3.4.1 Downsampling

Naturally, this architecture combines the methods used in both FIR and batch architectures to downsample. Where several control vectors are pushed to the shift-register at a slower rate. And the recursive calculations handle several vectors in parallel.

3.4.2 Two-stage downsampling

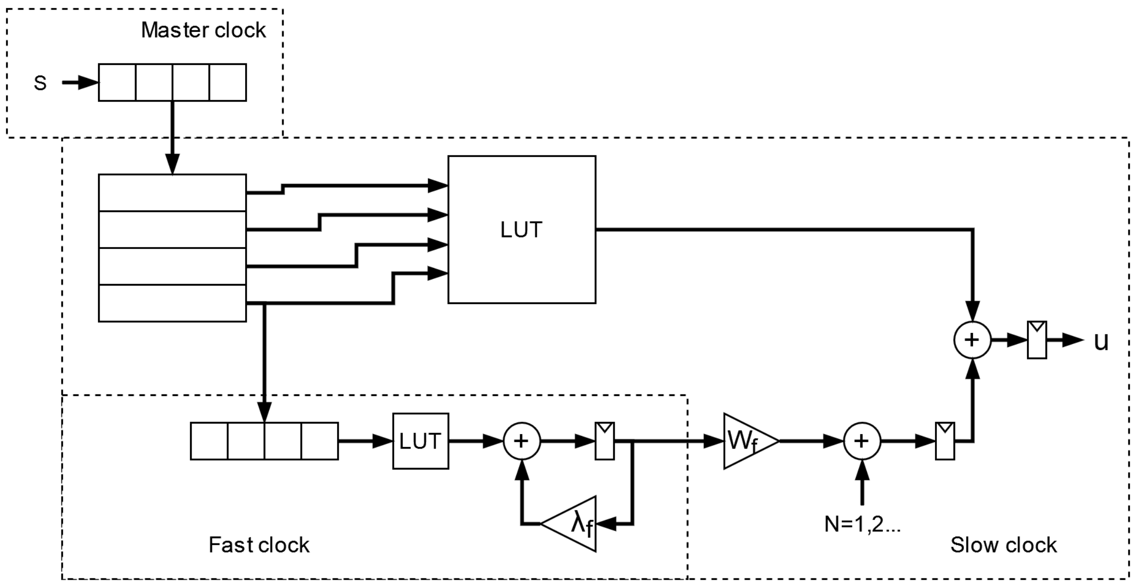


Figure 3.14: Hybrid two-stage implementation with system order N

Two-stage downsampling in the hybrid architecture only affects the lookback calculation. Its clock domains are shown in figure 3.14. The main shift-register still runs at the lowest frequency to minimize switching activity. The frequency is increased only in the recursions, so it uses another shift-register to split the control vector into smaller collections of vectors.

4 Results and Discussion

4.1 Toolchain

This toolchain was used to provide a compromise between the time used to generate a result, and the accuracy of said result. Since it was used on hundreds of configurations running PnR would be too time-consuming. Therefore a method of estimating power from a synthesized netlist was used.

The analog system was simulated using a python toolkit, an excerpt of the script used can be found in appendix A. A leapfrog system was used where a 1MHz sine wave was applied to its input. All filter constants were written to a SystemVerilog file since formatting text in python is simpler than importing data in SystemVerilog. Having a separate file for the constants makes changing the analog system simple. These constants were placed in a SystemVerilog package so they can be imported to any scope needed. This also means the constants could be defined as "localparam", which unlike "const" means the tools will consider them constant. Therefore they can be used to set parameters in a module. The control-vector stimuli were written to a comma-separated file, which can be read by the testbench.

The first step in the toolchain is register transfer level (RTL) simulation, which is done using Synopsys VCS. Here the functionality of the design is tested. The testbench used is as simplistic as it can be, only reading in stimuli and writing out sample results. It also generates a switching activity dump file of the design. The results from RTL simulations are plotted with a python script, where the PSD is plotted and SNR is calculated as well.

The second step is synthesis, which is done using Synopsys DC. It is performed in topographical mode, which means an approximate PnR is performed. This makes extracting parasitics for the wires possible, so the resistance and capacitance of all networks are calculated. This step generates the design's netlist, and the area estimate used in these results. Synthesis is performed using the SS corner of the technology library, this should give a worst-case scenario for timing.

This synthesis flow is mostly a standard topographical flow, but there are some special timing constraints used for the clock-dividers. For the clock-divider outputs, a "create_generated_clock" constraint is used where the clock division rate is set. This ensures the tools handle this network as a clock network and not as generic logic.

Finally, the power consumption was estimated using Primetime by Synopsys. Here the switching activity from RTL simulation and the design netlist and parasitics from synthesis were read. It also reads the technology library to use its info about the cells used.

At this step, the TT corner of the library was used to get a typical power consumption. Although, the library used was an "ultra-low threshold" library, which means it is extra fast and has a higher leakage current than the regular version. The application in this thesis does not need this high speed, so the library only leads to a higher static power than necessary. And it was not characterized at room temperature with the supply voltage used. Therefore the results will mostly focus on dynamic power, as this does not change as much. The worst-case leakage is used whenever it is included, which is the 125C node.

PnR was not a focus of this thesis. The scripts necessary to perform it on the 22nm technology were not ready. One PnR was run with the 28nm technology since the scripts were ready from a previous project. This was done because running an analog simulation on the netlist would be interesting to compare with the generated estimations. The results from that analog simulation showed the design using half as much dynamic power, which may be caused by the analog simulation not accounting for parasitics. When plotting the results from this simulation there appear to be some errors severely reducing the SNR, which can be seen in appendix C. It was theorized that it was a problem with negative slack as the synthesis was performed with the wrong clock frequency, but fixing that bug did not improve the simulation. Finding the cause of this discrepancy between digital and analog simulations ended up as future work.

4.2 SNR

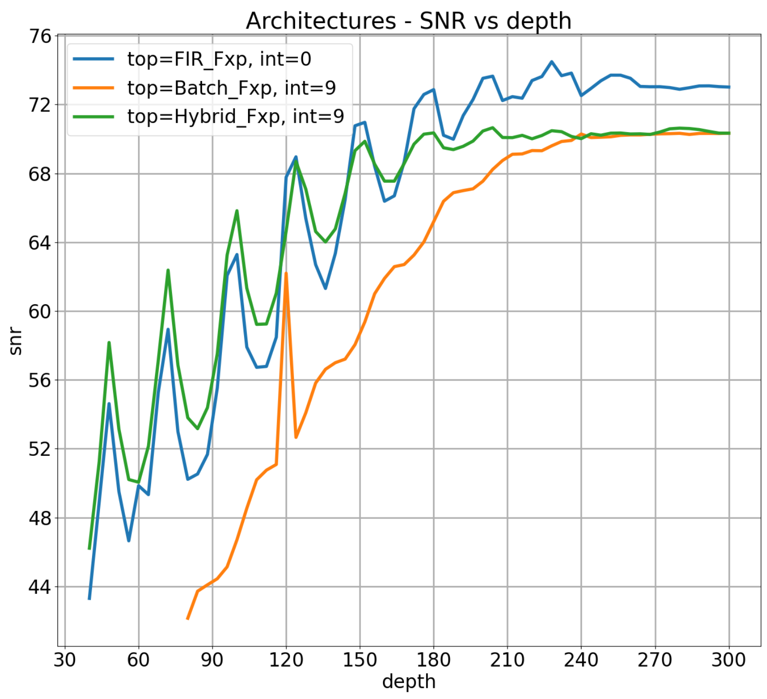


Figure 4.1: Basic architectures - SNR swept across filter-depth

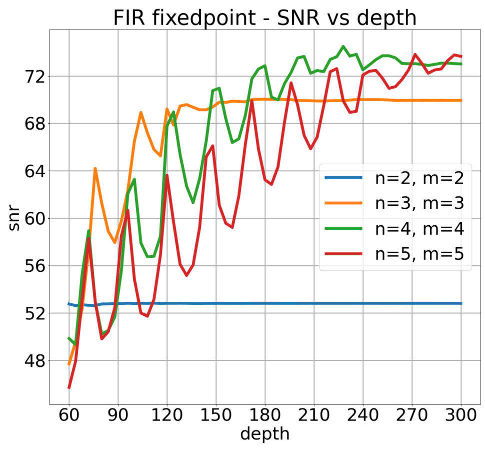
As seen in figure 4.1 the SNR of the FIR and hybrid architectures increase in waves as the filter depth rises. While the batch architecture rises more linearly. The figure also shows that the batch size needs to be larger than the filter depth of the other architectures to get the same SNR. The plot also shows the hybrid implementation's SNR generally higher than the FIR's. The FIR implementation crosses 60dB at a depth of 96, the hybrid implementation at 72, and the batch implementation at 160.

The SNR of the Hybrid and batch implementations are converging at a lower value than the FIR implementation does. This may be caused by the rounding mode used in the multiplication module, which always rounds down (toward $-\infty$). However, a simulation using the floating-point version of the batch implementation proved this false. The floating-point version should not have these rounding errors, but was still limited to the same SNR. It is not caused by an error in the transition between batches either, since the hybrid version is affected. There appears to be a minor error either in the implementation of these recursions, or in the constants they are using. Or possibly in one of the format conversions of these architectures.

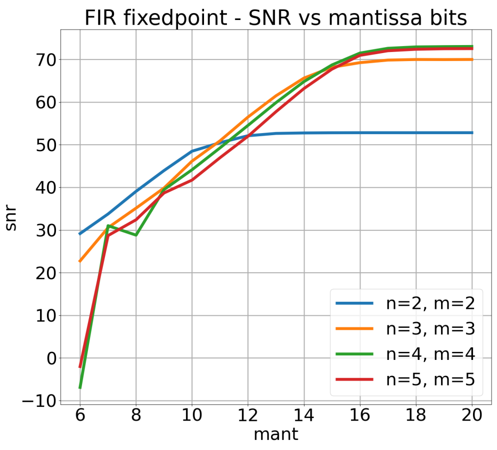
Figure 4.2a shows a plot of the SNR achieved by the FIR architecture across different filter depths. It shows that the SNR improves in waves as the filter depth increases. The plot also shows different system orders n. With a higher system order the peaks in SNR are more frequent. And with lower order, the SNR rises quicker while having a lower maximum. None of these configurations can reach more than 74dB due to the quantization noise caused by the number of bits the final results are limited to.

In figure 4.2b a sweep of SNR across the number of mantissa bits was performed. It shows that the SNR rises linearly, then flattens as it approaches its maximum. Across most of this range the different system orders are parallel, with approximately -2dB per increase in order.

A plot of SNR in a sweep of batch size is seen in figure 4.3a. In this architecture, the SNR does not have the valleys seen in the FIR architecture. And similar to the trend seen in figure 4.2a, a

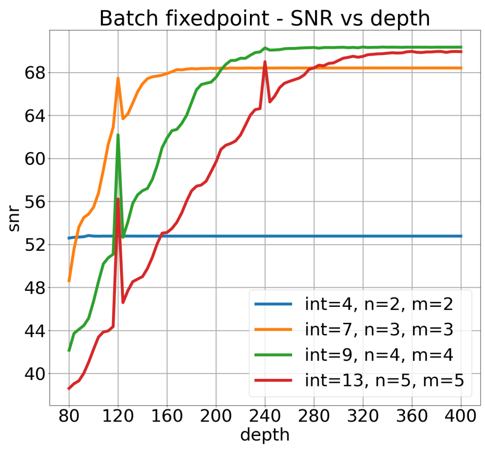


(a) SNR vs filter depth

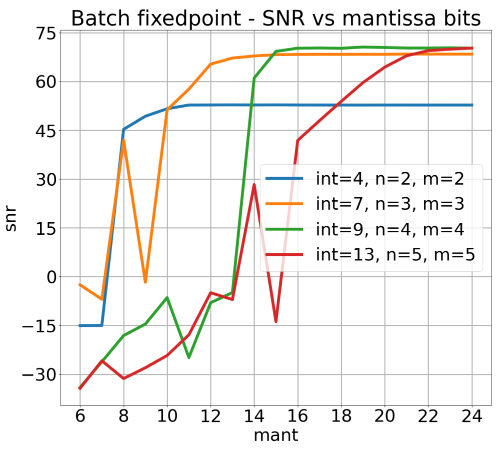


(b) SNR vs mantissa bits

Figure 4.2: FIR fixed-point - SNR using system order 2, 3, 4, and 5



(a) SNR vs batch size



(b) SNR vs mantissa bits

Figure 4.3: Batch fixed-point - SNR using system order 2, 3, 4, and 5

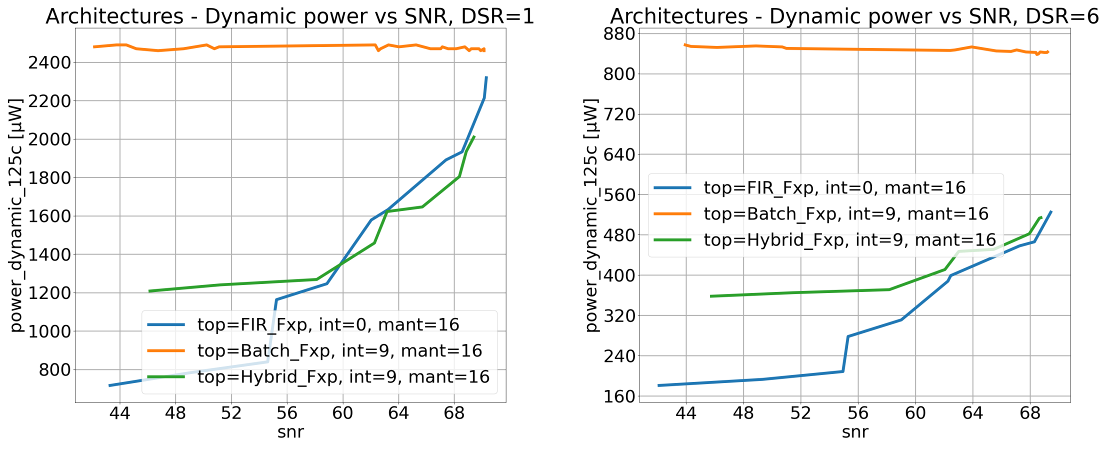
lower system order leads to a quicker rise in SNR. This plot also shows the batch implementation having a spike in SNR when the batch size is a multiple of 120. It is likely caused by the test-signal being 240 samples long, so the batch transitions can align with the zero crossing of the signal.

In figure 4.3b a plot of SNR across number of mantissa bits was made with the batch implementation. It shows that the SNR of the batch architecture has a steep increase, likely due to the recursions multiplying any errors. The system order has a much larger impact on the number of bits needed in the batch architecture than the FIR. It also requires more integer bits, which means a higher dynamic range is required in the recursions. The theory states that the state trajectories become larger when the results improve. This data supports that trend as the implementations requiring the lowest range are also limited to the lowest SNR.

Low precision causes an overswing in all implementations, which can lead to results overflowing. The FIR implementation is most impacted by filter depth, but the batch and hybrid implementations are most impacted by mantissa bits. Some examples of this can be seen in appendix B.

The hybrid implementation's trends follow the FIR trends for lookahead depth, but follow batch trends for mantissa bits. These figures are found in appendix D.

4.3 Filter depth



(a) DSR=1 (b) DSR=6

Figure 4.4: Basic architectures - Dynamic power swept across SNR

Figure 4.4a shows the dynamic power compared to the SNR achieved by these configurations. In practice, it is a sweep of filter depth or batch size normalized to the SNR achieved, because this gives a fairer comparison. The individual filter depth plots can be found in appendix E. This figure shows that the FIR implementation starts with lowest power, but grows fastest. Above 60dB SNR the hybrid implementation has lowest power. This is not a surprise since it should grow at half the rate FIR does. The estimates of the batch implementation do not include the RAM modules, so it only includes the control- and calculation logic. Which have no significant scaling with batch size. The batch implementation does however have the highest power consumption across the tested range. The RAM modules should increase in size linearly as batch size increases, their power consumption will come on top of the trend shown for the batch implementation.

However, the FIR implementation has the lowest power consumption across the entire range when DSR is increased, as seen in figure 4.4b. Its power consumption is still rising fastest, but the SNR required to surpass it is higher.

These results indicate that the FIR implementation is favored when the required SNR is low, or the DSR used is high. But there may be cases with higher SNR and low DSR where the hybrid or batch implementations use less power.

4.4 System order

Figure 4.5 plots how the architecture's dynamic power scale with the system order n. Increasing the system order will increase the number of bits in the control vectors, and also the number of analog state trajectories in the system. Every result gathered has used one digital control per analog state.

The figure shows the FIR implementation scaling mildly. The number of input bits in its LUTs would increase linearly, but since the filter coefficients are considered constant and nominal the synthesis tool can optimize the logic. This leads to the dynamic power flattening with high system orders. In contrast, the batch architecture scales faster. This is because the number of recursions needed increases in addition to the LUTs growing larger, resulting in a quadratic rise. Likewise, the lookback of the hybrid requires more recursions, while the lookahead scales linearly. Leading to the hybrid implementation's growth being between the other two; approximately a third of the rate batch scales plus half the rate FIR scales.

The size of the LUTs also depend on other factors. In the FIR implementation the size is determined

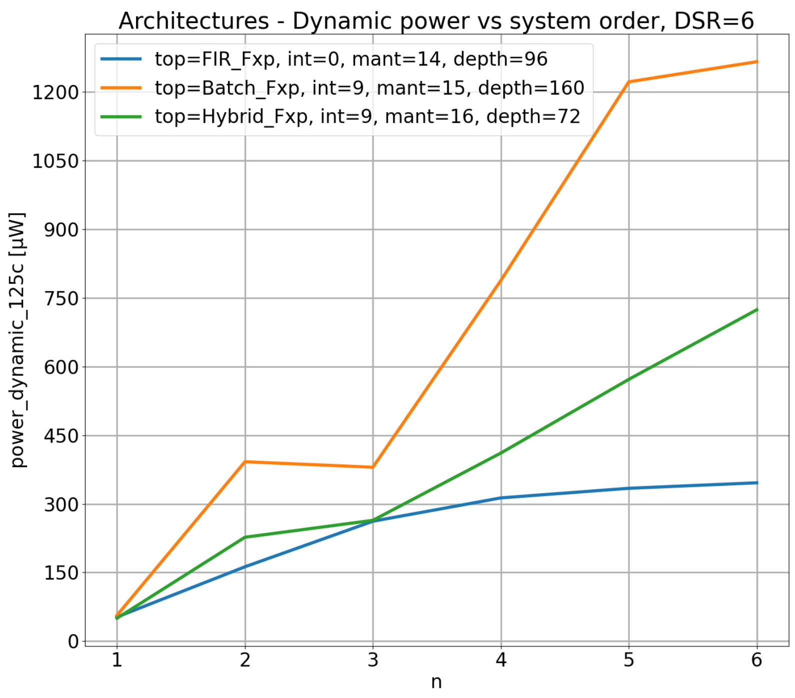


Figure 4.5: Basic architectures - Dynamic power swept across system order

by filter depth multiplied by control vector size. For the batch implementation, it is DSR multiplied by the control vector size. Different filter depths and DSRs will alter the rate of growth, but the trends should still look the same.

The area and total power show similar trends, their plots can be seen in appendix F.

Table 4.1: Optimized implementations with SNR above 60dB at 40MS/s, comparing system order 4 with order 3 and 6

Architecture	System order	Format	Depth	Area	Dynamic power	Total power
FIR	3	Q0.14	96	6887	262µW	1.34mW
	4	Q0.14	96	8677	313µW	1.66mW
	6	Q0.14	96	10587	346µW	1.99mW
Batch	3	Q7.12	124	11404	285µW	2.06mW
	4	Q9.15	160	30050	788µW	5.46mW
	6	Q15.22	292	103892	3.04mW	18.8mW
Hybrid	3	Q7.14	76	7358	227µW	1.39mW
	4	Q9.16	72	14630	411µW	2.72mW
	6	Q15.24	72	50938	1.32mW	9.29mW

As mentioned in section 4.2 changing the system order will also change the dynamic range required for the calculations. Table 4.1 shows a set of optimized configurations which all achieve 60dB SNR. A larger batch size was needed when system order increases, which means the lookahead-, and backward recursion needed more vectors to converge on a good estimate. However, the length of the FIR filter and the hybrid's lookahead did not require change. Although, section 4.2 shows that increasing the system order does reduce SNR in the FIR implementation as well. Increasing it further would require a deeper filter. The number of bits needed in the batch and hybrid implementations rises with system order, where a hybrid filter in a sixth-order system needs 40 bits. This secondary scaling exacerbates the trends shown in figure 4.5. Which works in the favor of the batch and hybrid implementations when system order is reduced. The table shows a hybrid filter in a third-order system having the lowest dynamic power.

These results show that all implementations are most efficient with a low system order. However, the FIR implementation changes less than the other implementations, making it more efficient when a high system order is used. When a low system order is used the hybrid implementation may be more efficient.

4.5 DSR

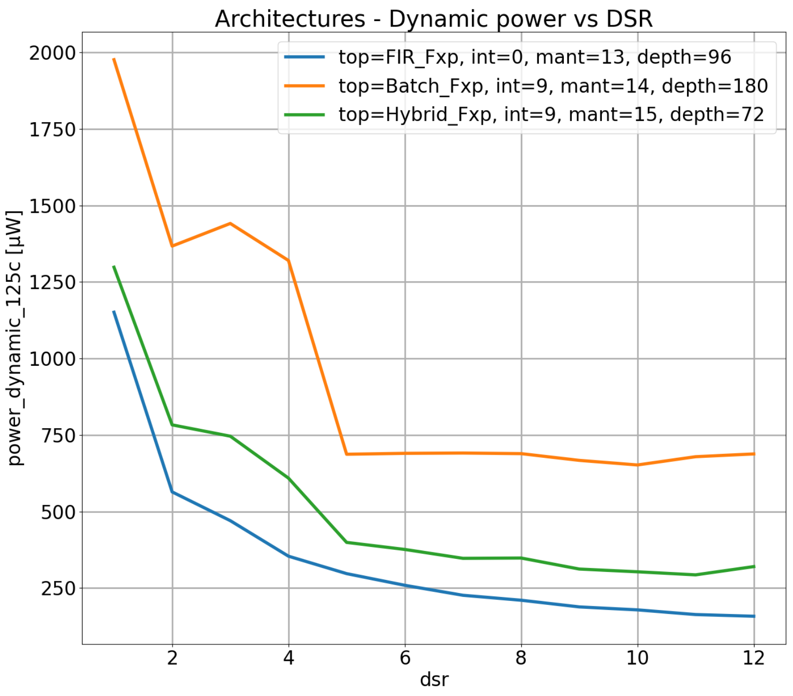


Figure 4.6: Basic architectures - Dynamic power swept across downsampling rate

Figure 4.6 shows a sweep over a range of DSRs using a configuration capable of 60dB SNR for each base implementation. The trend shown is that dynamic power sinks in all implementations as DSR is increased. The batch implementation seems to plateau above DSR=5, it is likely that its increased size is offsetting the effects of a lower frequency.

The total power consumption for the previous sweep is shown in figure 4.7. In the FIR architecture the power is still sinking since its area is barely changing. On the other hand, the batch architecture's power is rising with DSR, due to its LUTs becoming larger. However, the leakage power is very high with the technology node and temperature used here, so the effect of increased area is extra drastic. In the middle is the hybrid implementation which initially sinks, then rises above DSR=5 when the lookback size starts dominating the power estimates.

A plot of the area from this sweep is found in appendix G.

These results indicate that the FIR implementation is preferred when the DSR is high. Especially when the leakage power is expected to be high.

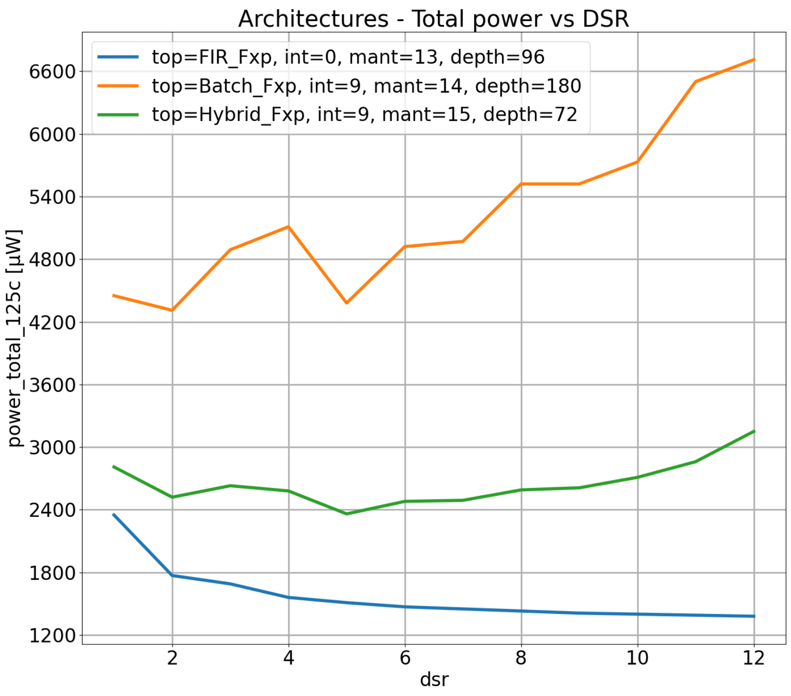


Figure 4.7: Basic architectures - Total power swept across downsampling rate

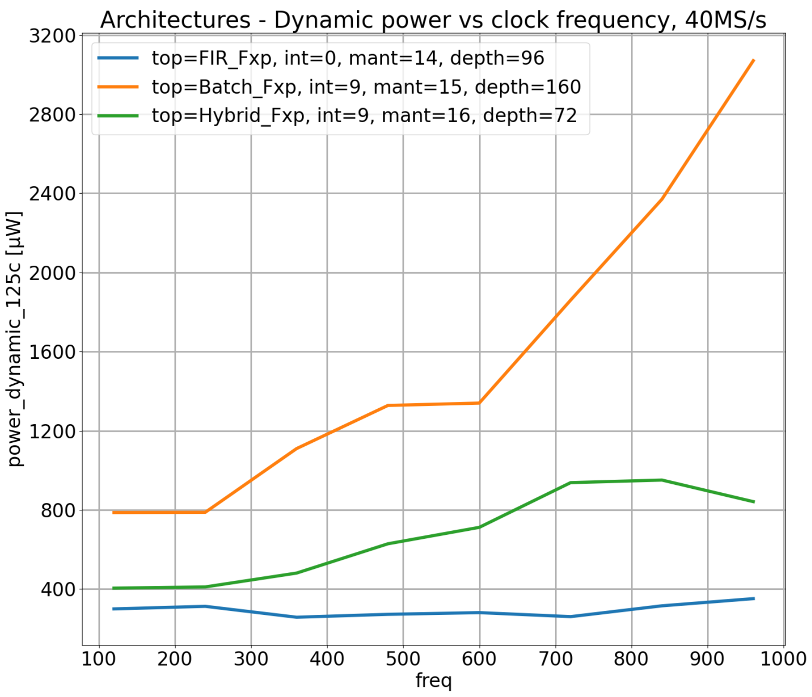


Figure 4.8: Basic architectures - Dynamic power swept across clock frequency, with constant output sample rate

4.6 Clock Frequency

Figure 4.8 shows the dynamic power consumption when the master clock frequency is changed. The output sample rate is kept at a constant 40MS/s in all these results, which means the DSR is adjusted in tandem with the frequency.

What the plot shows is that the power consumption of the batch- and hybrid architectures increase with a higher OSR, while the FIR architecture is not sensitive to this. Which is due to the area of both the batch and hybrid implementations scaling with DSR. In this case, the frequency of the recursions is not being reduced, leading to the dynamic power rising. The FIR implementation remains mostly unchanged across this plot, only the small input register becomes larger and runs faster. Therefore, it only has a slight increase in power consumption.

Plots of the total power consumption and area can be found in appendix H.

For a given application where the sample rate is set, this data suggests the OSR will not matter when using the FIR implementation. And a low OSR is preferred when the batch- or hybrid implementation is used.

However, the analog system changes when a higher frequency is used, giving different filter constants. This leads to the recursions needing a greater dynamic range at higher frequencies, and the FIR implementation needing a deeper filter. Which leads to an additional increase in area and power for all implementation.

Table 4.2: Optimized implementations with SNR above 60dB, comparing clock frequencies with 40MS/s output sample rate

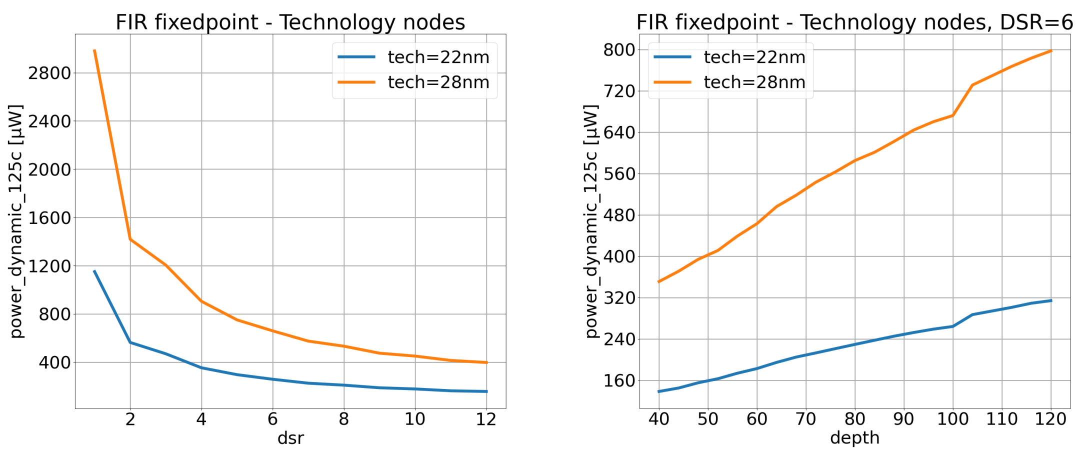
Architecture	Frequency	Format	Depth	Area	Dynamic power	Total power
FIR	240MHz	Q0.14	96	8677	313µW	1.66mW
	960MHz	Q0.14	184	11562	528µW	2.32mW
Batch	240MHz	Q9.15	160	30050	788µW	5.46mW
	960MHz	Q17.24	628	126978	4.91mW	23.3mW
Hybrid	240MHz	Q9.16	72	14630	411µW	2.72mW
	960MHz	Q17.19	96	30068	1.29mW	6.0mW

Table 4.2 shows configurations needed to achieve 60dB SNR with a clock frequency of 240MHz and 960MHz. The FIR filter depth nearly had to be doubled. The batch implementation needed 17 more bits in its data paths, and the batch size had to be four times as large. The Hybrid implementation needs 11 more bits and a slight increase in filter depth. This means that all implementations are more efficient when the frequency of the analog system is lower. Still, the FIR implementation has the smallest change of these implementations.

4.7 Technology node

To show the effect of using different technology nodes these plots were created. Figure 4.9a shows dynamic power swept across DSR using both the 22nm and the 28nm nodes. And figure 4.9b shows a similar sweep across filter depth. In these plots, the power consumption using the 28nm node is between two and three times higher than using the 22nm node.

All architectures show this trend, the plots for the other architectures can be found in appendix I. Even though this data can't be used to extrapolate power consumption in newer technology nodes it shows an improvement by scaling the technology node down.



(a) Dynamic Power vs DSR (b) Dynamic Power vs Filter Depth

Figure 4.9: FIR fixed-point - Dynamic power in 22nm and 28nm technology nodes

4.8 Two-stage downsampling

These results from the architectures with two-stage downsampling use a scatter-plot to show all possible combinations. Each point is marked with the two DSRs used where the first number is downsampling before recursions, and the second number is downsampling after. For example ”2/3” means the DSR before the recursion is two, and the DSR after is three. There is also a line plot showing the minimum values for the two-stage architecture.

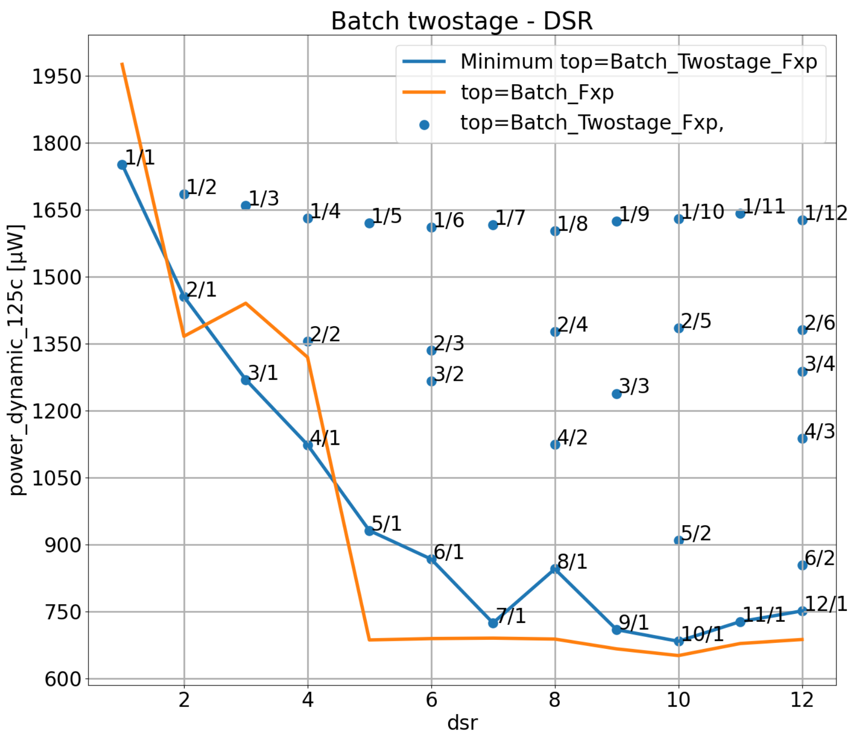


Figure 4.10: Batch Two-stage DSR - Dynamic power

Figure 4.10 shows the dynamic power consumption in the two-stage batch architecture compared to the regular batch architecture. The dynamic power is lowest when all the downsampling occurs before the recursions. Possibly because the logic becomes larger and the synthesis tool can perform more optimizations, and the reduced clock frequency should also contribute to this trend.

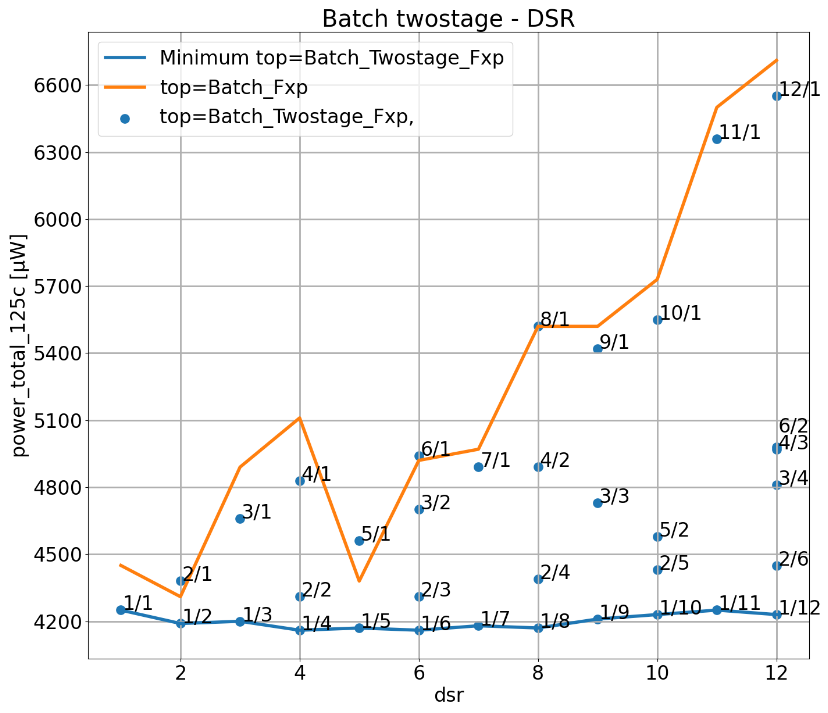


Figure 4.11: Batch Two-stage DSR - Total power

On the other hand, the total power is lowest when downsampling occurs entirely after the recursions, as seen in figure 4.11. The increased size when the recursions are downsampled increases the leakage power more than the dynamic power is reduced. But it also shows that downsampling after the recursions has no significant impact on power consumption in the batch architecture.

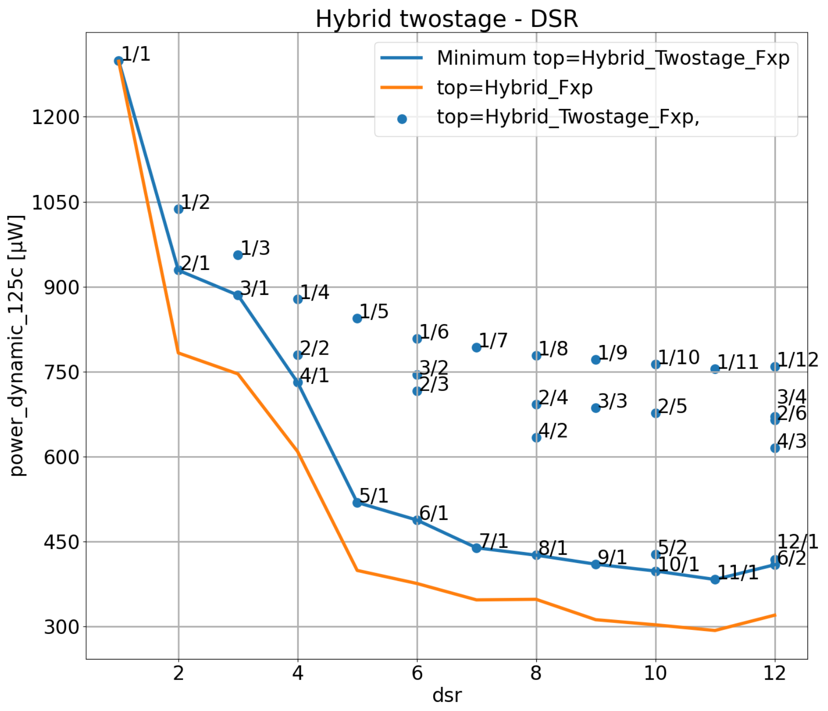


Figure 4.12: Hybrid Two-stage DSR - Dynamic power

A plot of the dynamic power in the hybrid architecture with two-stage downsampling is seen in figure 4.12. The trend is the same as the batch version, where downsampling early leads to the lowest dynamic power. It also shows that the regular hybrid implementation has lower dynamic power consumption than the two-stage. This is likely because the two-stage needs an additional clock divider and shift-register.

The total power consumption of the two-stage hybrid architecture is also lowest when downsampling late, shown in figure 4.13. This architecture does benefit from downsampling after the recursion because it reduces the frequency of the lookahead.

Plots of the area are found in appendix J.

The libraries used have excessive leakage power, so the effects of leakage power are overstated in these plots. Additionally, the temperature used in the estimations was 125 degrees celsius, giving the worst-case scenario for leakage. But, these two-stage architectures do fulfill their goal of reducing the size of these implementations. As seen in these plots a reduced area is beneficial if high leakage current is expected.

4.9 Staggered

Figure 4.14 shows the staggered FIR architecture with two, and three copies, compared with the standard version. The plot shows that there is no significant difference in dynamic power when DSR is higher than two. This means there is a lot of positive slack, so the synthesis tool can not optimize the design further. Resulting in no room for simplification in the staggered version. It does also show that two copies had the lowest power when there was no downsampling.

The synthesis tool reported 4ns slack in the critical path when DSR was 6, which is a lot when the clock period is 4.16ns. This critical path was located in the clock divider in all versions. When there was no downsampling the slack was 3.6ns in the staggered version with 2 copies, and 2.7ns in

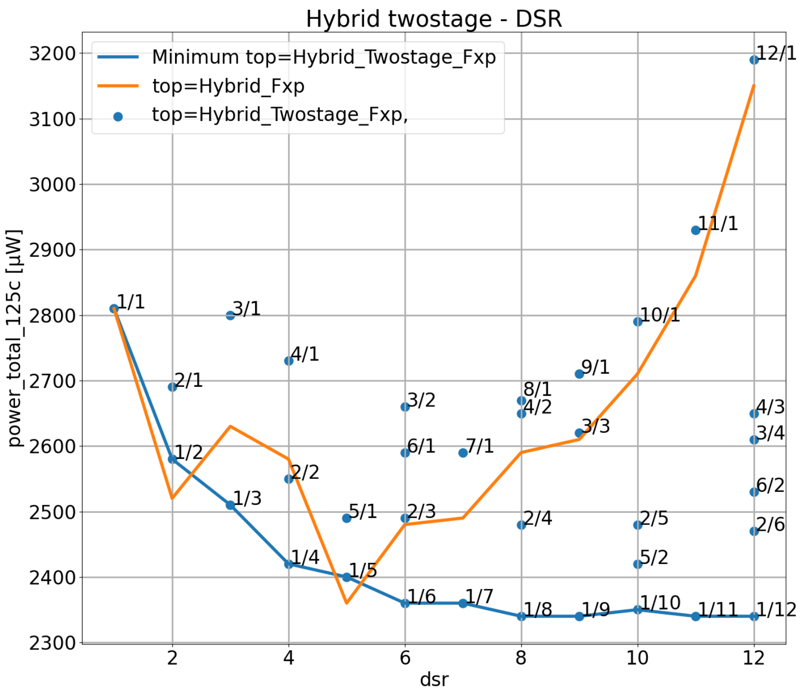


Figure 4.13: Hybrid Two-stage DSR - Total power

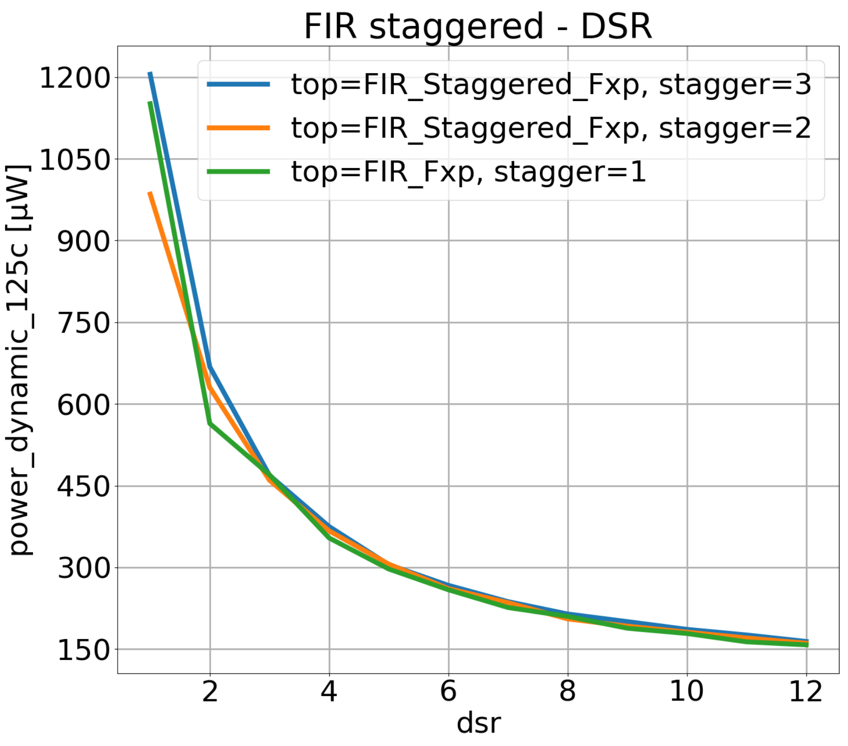


Figure 4.14: FIR staggered - Dynamic power vs DSR

the regular FIR. This means the staggered implementation can improve power consumption when slack is constrained.

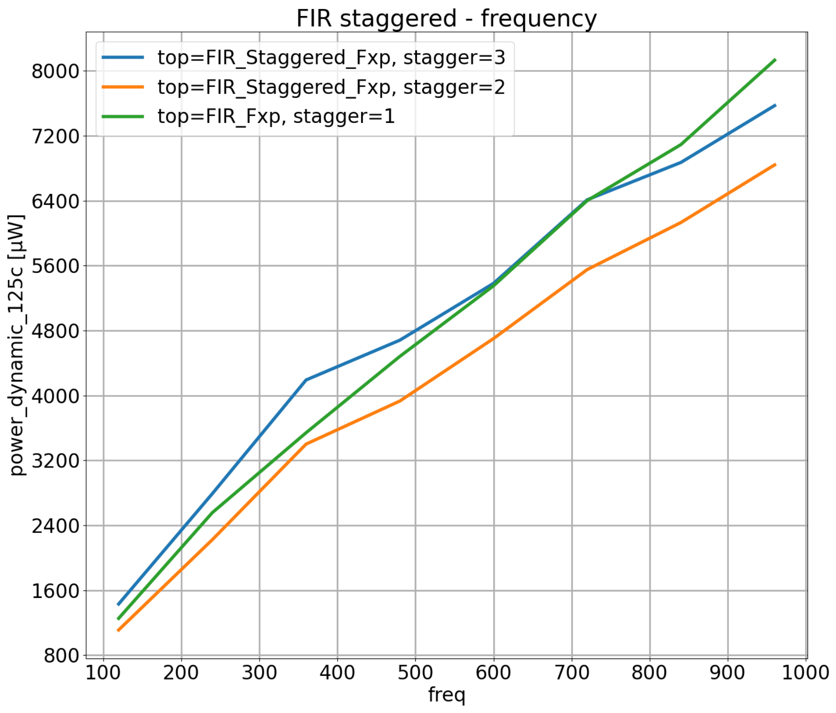


Figure 4.15: FIR staggered - Dynamic power vs Clock frequency

To better see the possible benefit of the staggered architecture a sweep across clock frequency was performed without downsampling, as seen in figure 4.15. This plot shows that the dynamic power of the regular version grows faster than both staggered configurations. Above 720MHz the regular FIR is using the most power, and the staggered version with two copies remain the lowest across the range.

All these results were gathered using the same supply voltage, but this strategy works best when the supply voltage is reduced. Using more copies means the calculation circuit is running slower. Since supply voltage is linked to the speed of circuits it means a slower circuit can use a lower voltage, which can reduce power consumption drastically.

The findings in figure 4.15 indicate that the staggered implementation can be more efficient when the timing is constrained. This will be the case when the supply voltage is minimized.

4.10 Floating-point

Table 4.3: Floating-point and fixed-point implementations with SNR above 60dB, using DSR=6

Architecture	Format	Depth	Area	Dynamic power	Total power
FIR fixed-point	Q0.14	96	8677	313µW	1.66mW
FIR floating-point	F5.10	96	51534	2.64mW	11.3mW
Batch fixed-point	Q9.15	160	30050	788µW	5.46mW
Batch floating-point	F6.9	160	55108	2.0mW	11.1mW

Floating-point configurations have not been a focus in this work, as previous work [4] with these filters indicated they required a lot more resources than fixed-point. Table 4.3 shows a collection of floating-point and fixed-point configurations with 60dB SNR. This data shows that the floating-point FIR is five times larger, and uses ten times more power, than its corresponding fixed-point version. The floating-point batch architecture used less power than the FIR, but still was roughly twice as resource intensive as its fixed-point version.

Floating-point arithmetic is more complex than fixed-point, with floating-point addition being the most complex. This makes it harder for the synthesis tool to optimize and combine logic. Which has a severe impact on the FIR implementation because its LUTs use many adders. The positive side of floating-point numbers can be seen in the batch version, where the number of bits needed to achieve the target SNR is lower than in the fixed-point version. The numbers in the recursions benefit from an increase in dynamic range. It is possible that the batch implementation could be more efficient if it used fixed-point arithmetic in the LUTs, and floating-point arithmetic in the recursion. Especially with higher system orders where the state vectors need a higher range.

4.11 Delay in implementations

The delay of each implementation can be summarized with some simple formulas, shown in table 4.4. Startup delay is the time from a reset occurs until the first valid sample is output. Throughput delay is the delay from a sample is taken until its value is calculated during regular operation. All these delays are given in a number of clock cycles, so to calculate the time delay it just needs to be divided by the master clock's frequency.

Table 4.4: Summary of delay in architectures

Architecture	Startup delay	Throughput delay
FIR	$L_{tot} + DSR * (LUT_d + 2)$	$L_b + DSR * (LUT_d + 2)$
FIR staggered	$L_{tot} + DSR * (LUT_d + 2 * Stagger)$	$L_b + DSR * (LUT_d + 2 * Stagger)$
Batch	$4 * L_{bat} + DSR * (LUT_d + 4)$	
Batch two-stage	$4 * L_{bat} + DSR * 2 + DSR_1 * (LUT_d + 2)$	
Hybrid	$L_b + DSR * (LUT_d + 2)$	
Hybrid two-stage		

In table 4.4, L_{tot} is the total filter length. That is, both forward- and backward length. L_b is only the lookahead length, while L_{bat} refers to the batch size used. LUT_d is delay of the LUTs, which is elaborated below in equation 4.1. "Stagger" in the table refers to the number of copies used in the staggered implementation.

Startup delay only counts clock cycles until all registers are filled with data. Since the hybrid and batch architectures use an infinite lookback this will happen at the same pace as the throughput delay. However, it may take longer before the lookback estimates are accurate.

The two-stage batch implementation has a slightly lower delay than the regular batch implementation. This is because the recursions are using a faster clock. The two-stage hybrid implementation is not affected since the lookahead is constraining its delay. The staggered FIR implementation has more delay than the regular due to its additional shift-registers.

$$LUT_d = log_2(\frac{LUT_{width}}{LUT_{max\_size}})/LUT_{comb} \tag{4.1}$$

The delay of a LUT is shown in equation 4.1. LUT_{max_size} refers to the size these LUTs are broken down to when they are too large. The implementations have used a LUT size of 4 when fixed-point is used, and 6 when floating-point is used. The formula finds how many layers of adders are used, and divides it by how many layers are between each set of registers. All implementations have used three layers of combinational adders between registers. The width of a LUT is how many bits are on its input, which in FIR is filter length multiplied by control-vector size, and in recursions is DSR multiplied by control-vector size. In the hybrid implementation, the lookahead is typically slowest, so the LUT delay is calculated the same way as FIR.

The table indicates that filter depth or batch size is the primary source of delay, and that LUTs and DSR contribute a lesser amount of delay. It also shows that batch size has a greater impact on delay than filter depth does.

5 Future Work

So far the work on estimation filters has been purely theoretical, hopefully, an attempt at a physical realization will be performed in the future. Then some real-world results could be gathered. However, before the design is ready it has to be verified to ensure it will work as expected. There has been performed an analog simulation on a design. A configuration of the FIR implementation was placed and routed to be simulated on, but the results from analog simulation did not match the results from digital simulation. That issue ended up outside the scope of this thesis but would need to be investigated further before a physical version is made. Also, running simulations with realistic stimuli is likely a good idea to detect potential problems. For example, the assumption made in this thesis of no digital anti-alias filtering being necessary should be verified.

The implementations in this thesis have assumed that all filter constants were nominal, while this won't be the case in reality. A method of calibrating these values after fabrication should be chosen. Making the filter constants adjustable will likely reduce the amount of optimization the synthesis tool can do, so it will have a negative impact on power consumption. But, the extent of calibration needed is also unknown, perhaps changing a couple of the LSBs would be enough. This would depend on the variation in the analog system. So evaluating the amount of calibration needed, and its impact on power consumption remains.

There are still more low-power optimizations to explore. A multiple supply voltage system would likely improve power consumption in the staggered architecture, and in general when down-sampling. It is possible that an estimation filter with a reduced SNR followed by a conventional digital filter could be more efficient than just an estimation filter. And there are likely a lot of clever optimizations to evaluate.

This thesis has only evaluated SISO architectures of the estimation filter. Meanwhile, ultrasound imaging is a multiple-in-multiple-out (MIMO) application, so implementations for this should be explored in the future. Also, in chapter 10.4 of [1] the possibility of resource sharing in a control-bounded ADC with multiple inputs is mentioned. That is, an ADC with multiple inputs could have inputs sharing analog states. This would be interesting since sharing analog states could put the FIR implementation at a disadvantage, as it would have to replicate the entire filter per output.

6 Conclusion

This thesis has explored implementations of estimation filters for control-bounded ADCs with a single input. Providing estimates for power and area, which hopefully can be used to decide parameters of the overall system in the future. While these results are still only estimated from simulations, they should be more accurate than earlier estimates.

Three working implementations have been described, and a variation of each was proposed. The implementations were written in SystemVerilog and are freely available on GitHub [5].

The trends shown by the FIR implementation are a large increase in both area and power with filter depth, which is the primary factor determining SNR. When the DSR is increased the size of this implementation shows minimal change, but its power consumption sinks. The system order has a linear relation to size and power consumption, the rate of this change depends on the filter depth. The FIR implementation is not sensitive to OSR as long as the output sample-rate is fixed. Although, it will likely need a deeper filter when OSR rises, which increases its size and power consumption.

In the batch implementation, the trends show no significant increase in size or power with batch size/SNR, but this is because the RAM modules have been excluded from estimations. Still, its power consumption is higher than other implementations in the tested range (less than 70dB SNR). The size of this implementation increases when DSR is higher, while the power became lower before plateauing. The leakage in the technology will have a great impact on this trend since the area is increasing. This implementation shows the largest increase in both area and power when the system order is increased. When increasing the OSR, and keeping the output sample rate fixed, both the power consumption and size of this implementation will increase.

The hybrid implementations' trends are usually between the other two. Its area and power scales slower with filter depth than FIR does, achieving the lowest power consumption above 60dB when the DSR is one. The area of this implementation increases with DSR, but at a slower rate than the batch implementation. The power decreased even when including the high leakage of the estimations, but it eventually starts rising as DSR is increased further. When the system order increases its area and power rise at a rate between the other two. With a fixed output sample rate and increasing OSR both the area and power consumption rises.

The implementations using two-stage downsampling were made to reduce the size when downsampling, which they did succeed at. Whether they reduced the power consumption depends, as it did improve the total power consumption in these estimations. However, their leakage power was high due to the technology library and temperature used. This means the regular version will be more efficient when low leakage is expected.

The staggered implementation aimed to reduce the power of the FIR implementation further, but the libraries available were not characterized for low enough voltages to fully explore this. It did show better efficiency than the regular implementation at high speeds, which indicates a potential improvement when the voltage is lowered.

While there is still a lot of work to do in this field of ADCs, this thesis has hopefully provided some useful information on the estimation filter's power consumption and area. These findings indicate that the FIR implementation is best suited for this application, achieving a dynamic power of 313µW and area of $8677\mu m^2$. Alternately, the hybrid implementation in a third-order system was lower. It achieved 227µW dynamic power with an area of $7358\mu m^2$. While this power consumption is higher than the target, it can likely be improved further by using a lower supply voltage.

Bibliography

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

[2] F. E. Feyling, 'Design considerations for a low-power control-bounded a/d converter', M.S. thesis, 2021. [Online]. Available: https://hdl.handle.net/11250/2824253 (visited on 12th Dec. 2021).

[3] D. A. B. Mikkelsen, 'Implementing a digital estimator for a control-bounded adc', NTNU, Tech. Rep., 2021.

[4] ——, 'Analyzing estimation filter implementations for control-bounded adcs', NTNU, 2021.

[5] ——, 'Hdl source code'. (2022), [Online]. Available: https://github.com/GuavTek/Control_bounded_filter_HDL/ (visited on 18th Jun. 2022).

[6] M. D. Ercegovac and T. Lang, *Digital Arithmetic.* Morgan Kaufmann, 2004, pp. 397–417, ISBN: 9781558607989. [Online]. Available: https://search.ebscohost.com/login.aspx?direct=true&db=nlebk&AN=210506&site=ehost-live&scope=site.

[7] R. Woods, J. McAllister, G. Lightbody and Y. Yi, *FPGA-based Implementation of Signal Processing Systems.* Wiley, 2008, pp. 38–55, ISBN: 9780470030097. [Online]. Available: https://search.ebscohost.com/login.aspx?direct=true&db=nlebk&AN=250685&site=ehost-live&scope=site.

[8] S. Stevenson, 'Rounding in fixed point number conversions', 2009. [Online]. Available: https://sestevenson.wordpress.com/2009/08/19/rounding-in-fixed-point-number-conversions/ (visited on 8th Jun. 2022).

[9] T. C. Carusone, D. Johns and K. Martin, *Analog Integrated Circuit Design, 2nd edition, international student version.* John Wiley & Sons, Inc., 2012, pp. 543–544, 609–612, 696–719, ISBN: 978-1-118-09233-0.

[10] P. R. Panda, A. Shrivastava, B. V. N. Silpa and K. Gummidipudi, 'Basic low power digital design', in *Power-efficient System Design.* Boston, MA: Springer US, 2010, pp. 11–39, ISBN: 978-1-4419-6388-8. DOI: 10.1007/978-1-4419-6388-8_2. [Online]. Available: https://doi.org/10.1007/978-1-4419-6388-8_2.

[11] D. A. B. Mikkelsen. 'Python scripts used in this project'. (2022), [Online]. Available: https://github.com/GuavTek/control_bounded_filter_models/ (visited on 18th Jun. 2022).

Appendix

Appendix A - A snippet of python code used to generate the analog system's constants, and the stimuli used in simulations.

Appendix B - A selection of plots from RTL simulation.

Appendix C - A plot of the FIR implementation after an analog simulation.

Appendix D - Plots of the SNR trends in the hybrid architecture.

Appendix E - Individual depth sweeps for each of the basic implementations.

Appendix F - Plots of total power consumption and area swept across system order.

Appendix G - An area plot of the DSR sweep

Appendix H - Area and total power plots from the frequency sweep.

Appendix I - Additional plots showing differences between 22nm and 28nm technology nodes.

Appendix J - Area plots of the two-stage downsampling architectures.

Appendix K - Area and dynamic power plots for a sweep of the number of mantissa bits.

Appendix L - A failed implementation included from a previous work [4].

A Analog system

The analog system was simulated with the cbadc python package to generate coefficients and stimuli for these simulations.

The following python code instantiates the analog system used in this project, and formats the filter constants and stimuli vectors as needed. The full script can be found on GitHub [11].

```python
import cbadc
import numpy as np

N = 4                               # Analog states
M = N                               # Digital states
samples_num = 24000                 # Length of generated stimuli
FIR_size = 400                      # Number of coefficients generated
Fs = 240e6                          # ADC sampling frequency
Fc = 5e6                            # Filter cut-off
amplitude = 0.95                    # Input signal amplitude
Fi = 1e6                            # Input signal frequency
Ts = 1.0 / Fs                       # ADC sampling period
end_time = Ts * samples_num         # Simulation end

kappa = -1.0
beta = 1.0 / (2 * Ts)
Wc = 2*np.pi*Fc
rho = -Wc**2/(4*beta)
betaVec = beta * np.ones(N)
rhoVec = np.array([0 if i==0 else rho for i in range(N)])
gammaVec = kappa * beta * np.eye(N)
G_at_omega =
↪  np.linalg.norm(analog_system.transfer_function_matrix(np.array([Wc])))
eta2 = G_at_omega**2

# Instantiate a leapfrog analog system.
analog_system = cbadc.analog_system.LeapFrog(betaVec, rhoVec, gammaVec)

# Initialize the digital control.
digital_control = cbadc.digital_control.DigitalControl(Ts, M)

# Initialize estimator
digital_estimator = cbadc.digital_estimator.ParallelEstimator(analog_system,
↪  digital_control, eta2, samples_num)

# Instantiate FIR estimator
L1 = FIR_size
L2 = FIR_size
FIR_estimator = cbadc.digital_estimator.FIRFilter(analog_system, digital_control,
↪  eta2, L1, L2)

# Instantiate the analog signal
analog_signal = cbadc.analog_signal.Sinusodial(amplitude, Fi)

# Instantiate the simulator.
simulator = cbadc.simulator.StateSpaceSimulator(analog_system, digital_control,
↪  [analog_signal], t_stop=end_time)



```

```
# Extract filter constants
Lf = digital_estimator.forward_a
Ff = digital_estimator.forward_b
Lb = digital_estimator.backward_a
Fb = digital_estimator.backward_b
Wf = np.reshape(digital_estimator.forward_w, M)
Wb = np.reshape(digital_estimator.backward_w, M)

h1 = FIR_estimator.h[0][0:FIR_size]
h2 = FIR_estimator.h[0][FIR_size:]

hb = h2
hf = np.zeros((FIR_size, M))
for i in range(0, FIR_size):
    hf[i] = h1[FIR_size-i-1]

# Format stimuli
tVectors = np.zeros((N, samples_num), int)
x = 0
for s in simulator:
    y = 0
    for num in s:
        if (num == True):
            tVectors[y][x] = 1
        elif (num == False):
            tVectors[y][x] = -1
        y += 1
    x += 1

# tVectors is saved in a .csv file
# filter constants are written to a .sv file in a SystemVerilog package
```

B RTL simulation plots

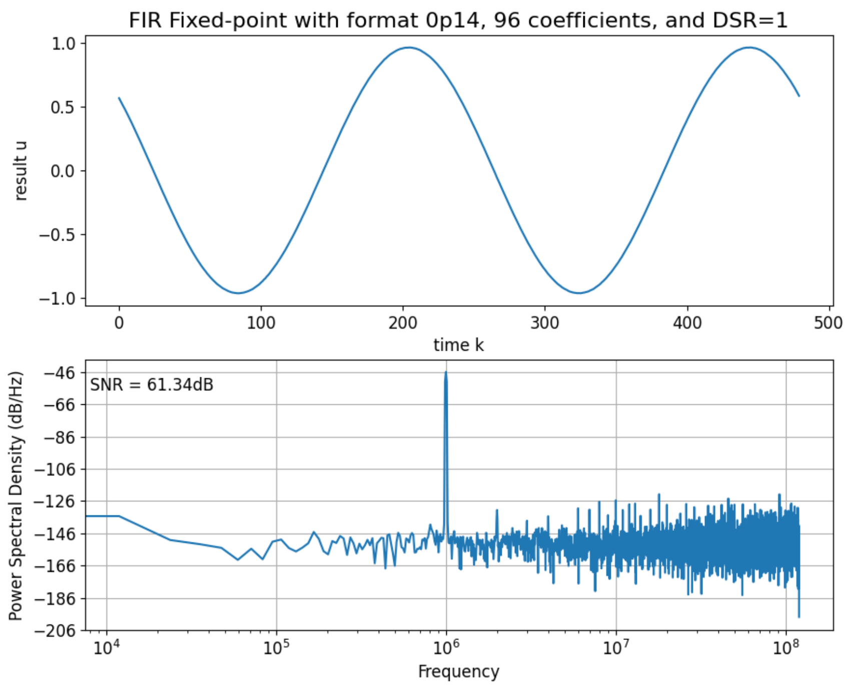


Figure B.1: FIR fixedpoint - RTL simulation

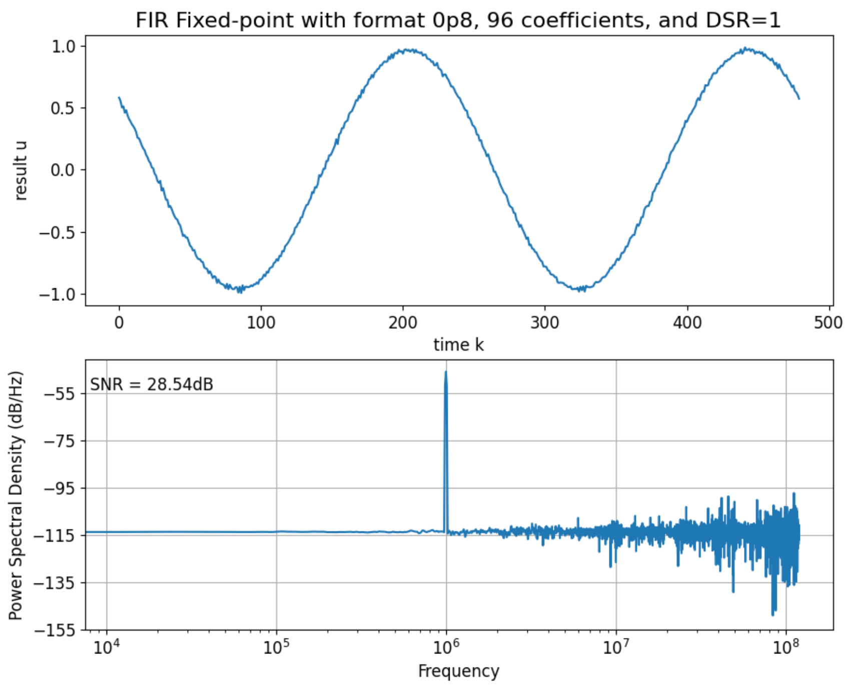


Figure B.2: FIR fixedpoint - RTL simulation with few mantissa bits

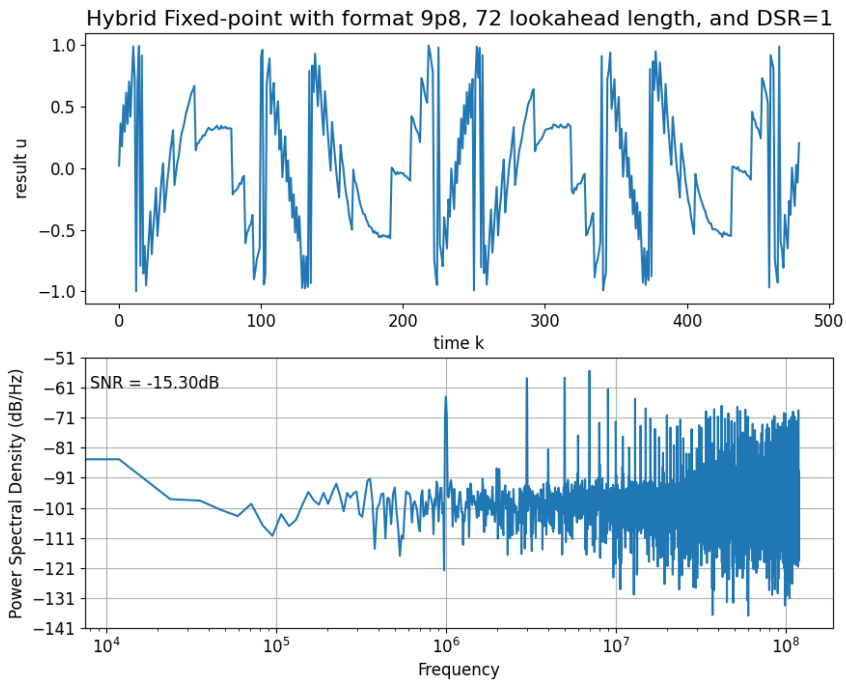


Figure B.3: Hybrid fixedpoint - RTL simulation with too few mantissa bits

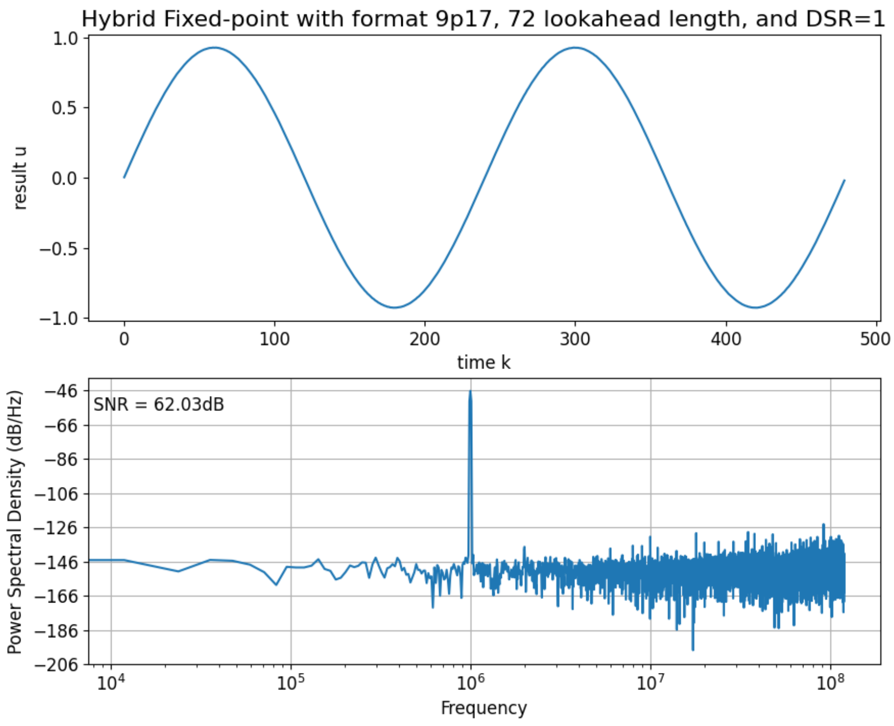


Figure B.4: Hybrid fixedpoint - RTL simulation

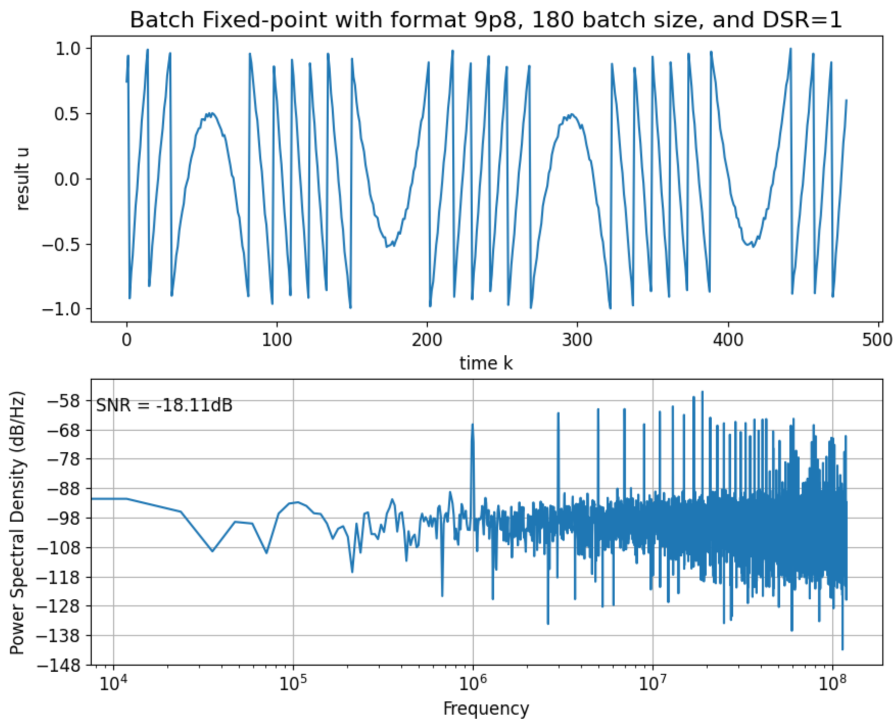


Figure B.5: Batch fixedpoint - RTL simulation with too few mantissa bits

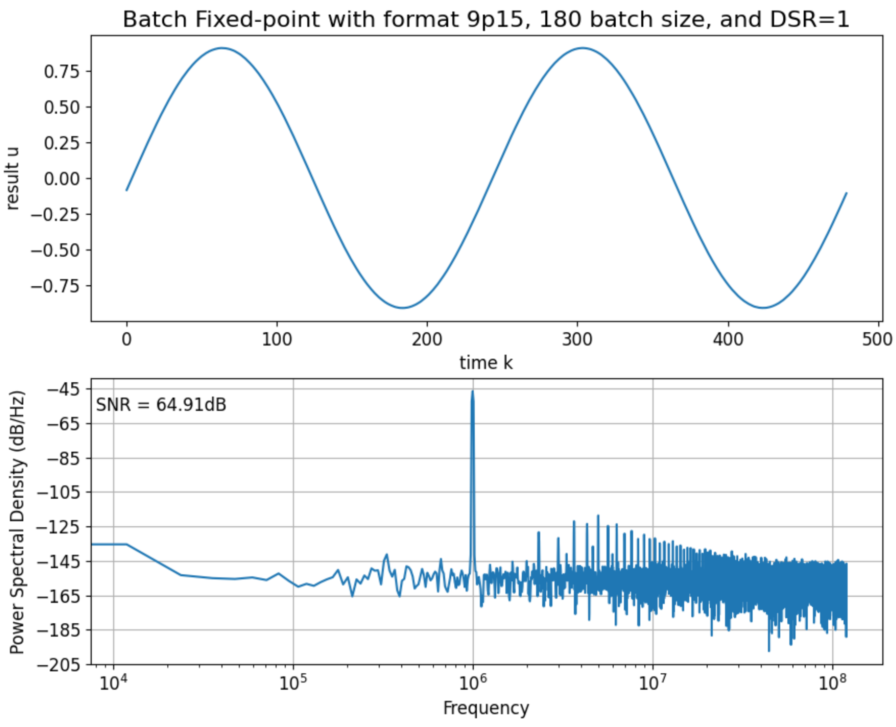


Figure B.6: Batch fixedpoint - RTL simulation

C Post PnR simulation plots

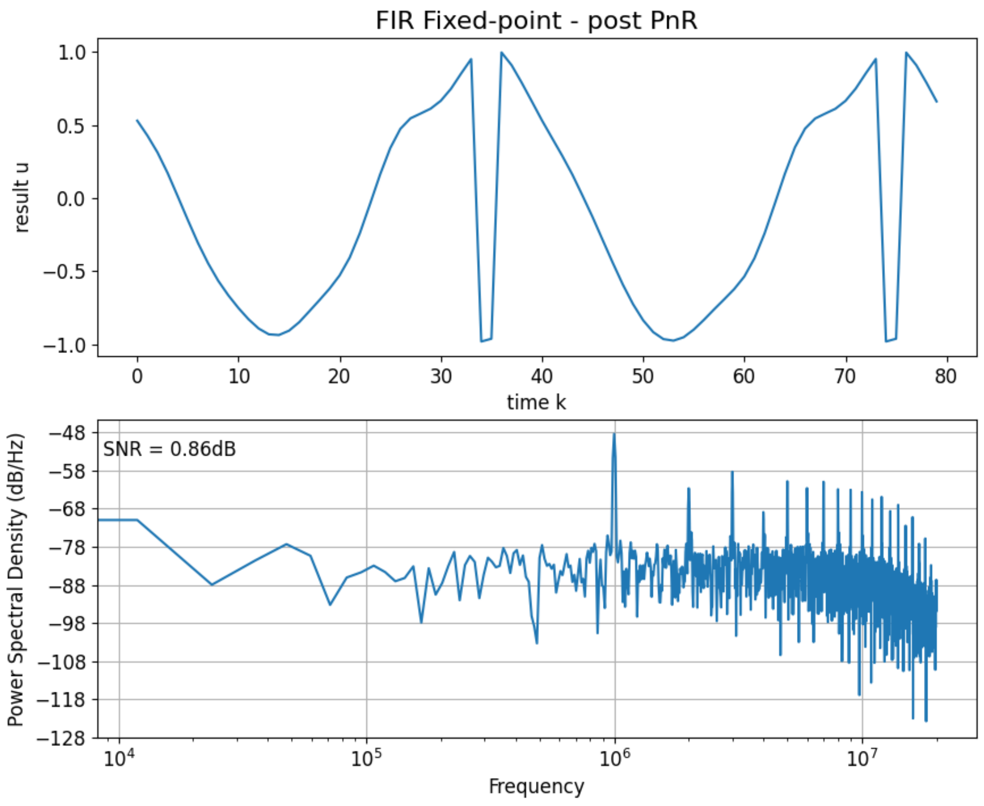


Figure C.1: FIR fixedpoint - Spectre simulation after PnR

D SNR plots

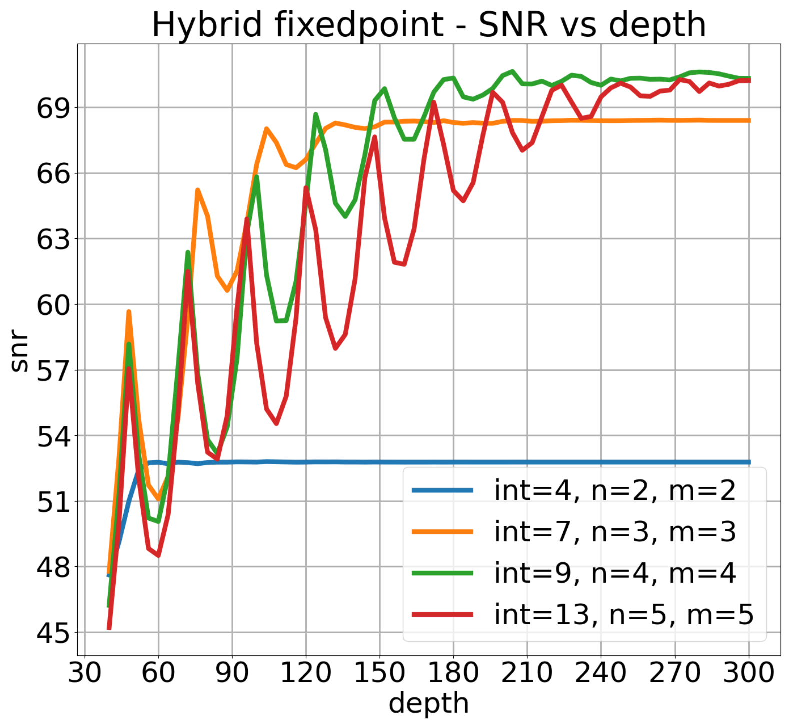


Figure D.1: Hybrid fixedpoint - SNR vs Filter depth using different system orders

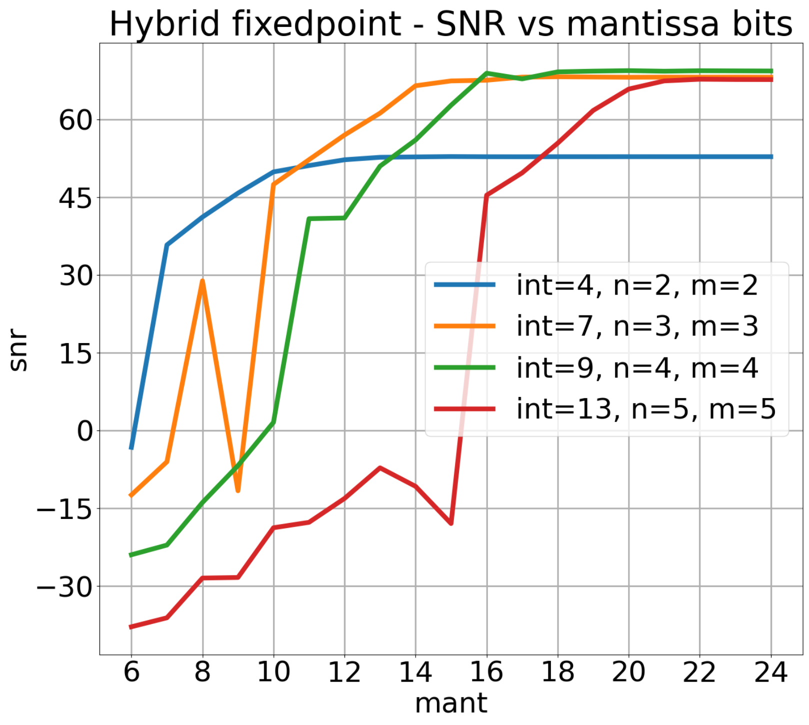


Figure D.2: Hybrid fixedpoint - SNR vs Mantissa bits using different system orders

E Depth plots

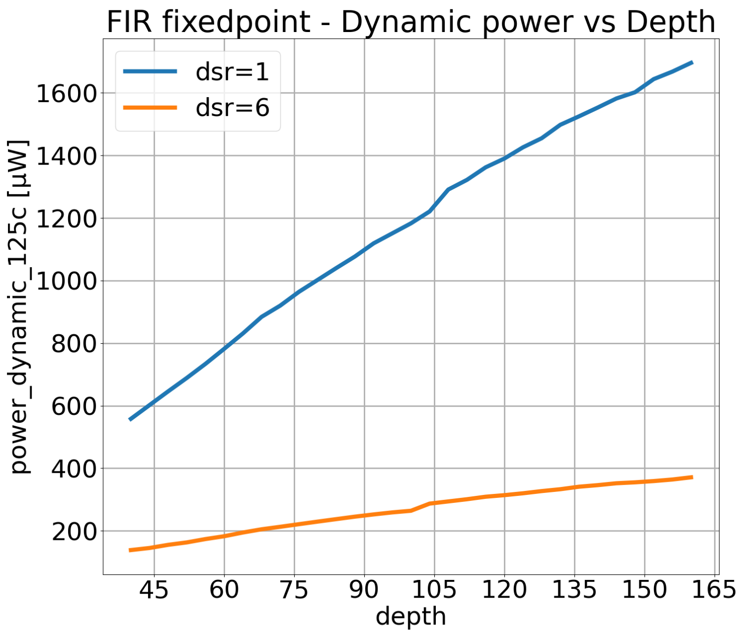


Figure E.1: FIR fixedpoint - Dynamic power vs Filter depth

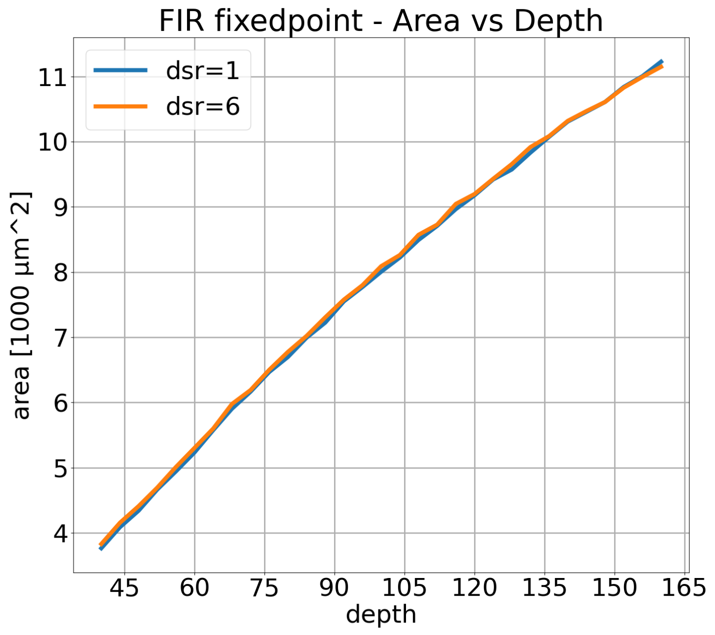


Figure E.2: FIR fixedpoint - Area vs Filter depth

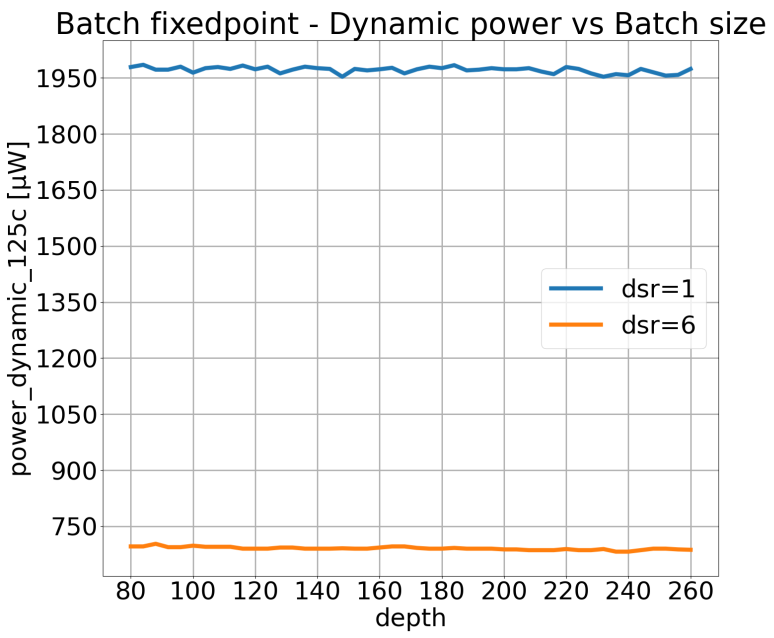


Figure E.3: Batch fixedpoint - Dynamic power vs Filter depth

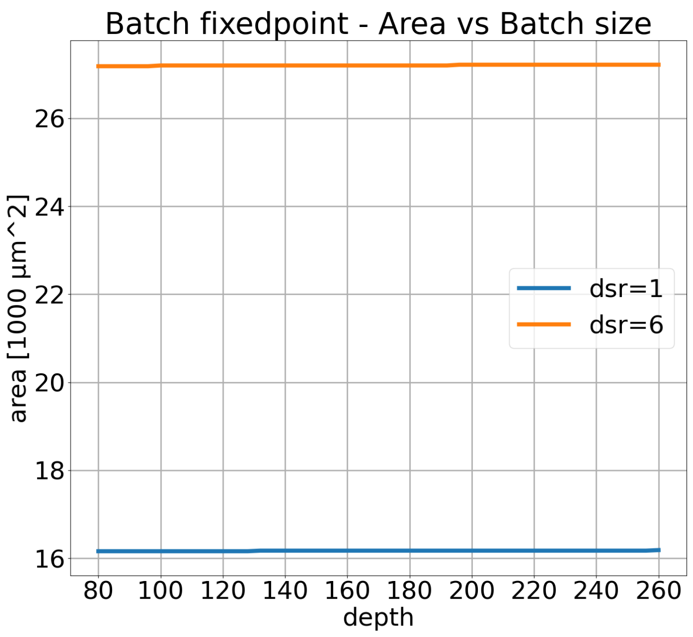


Figure E.4: Batch fixedpoint - Area vs Filter depth

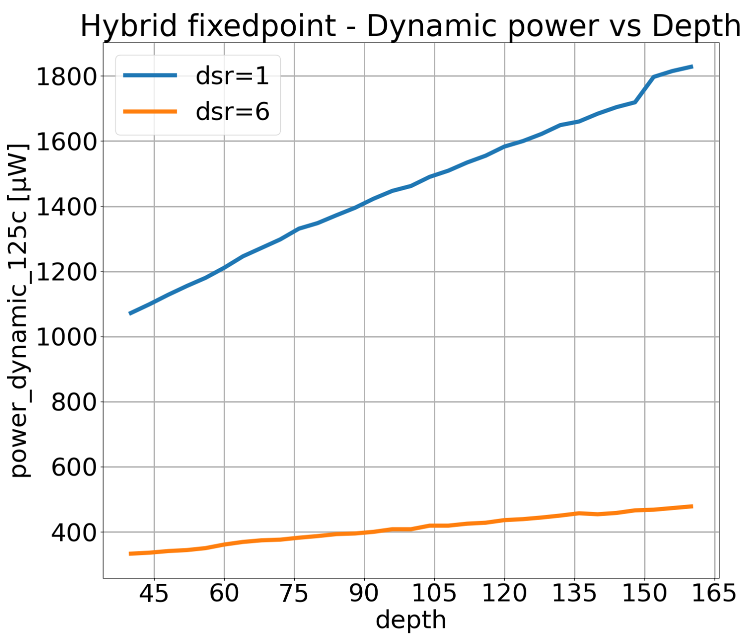


Figure E.5: Hybrid fixedpoint - Dynamic power vs Filter depth

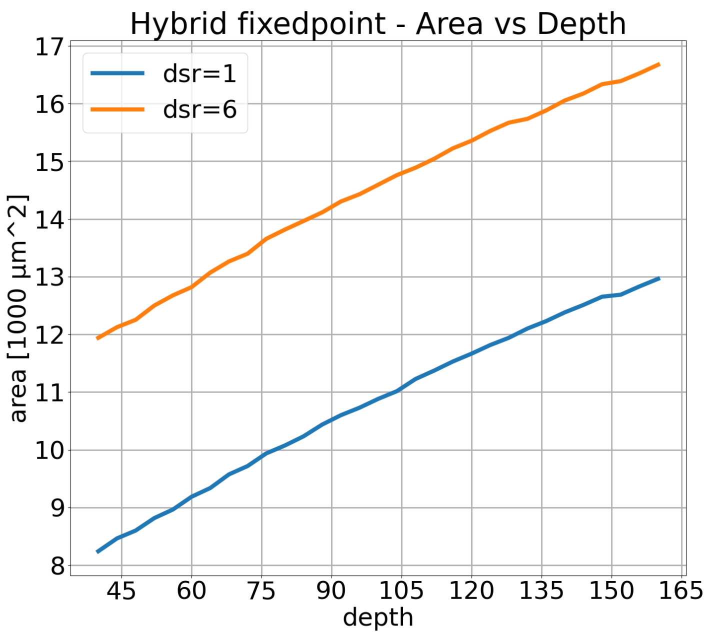


Figure E.6: Hybrid fixedpoint - Area vs Filter depth

F System order plots

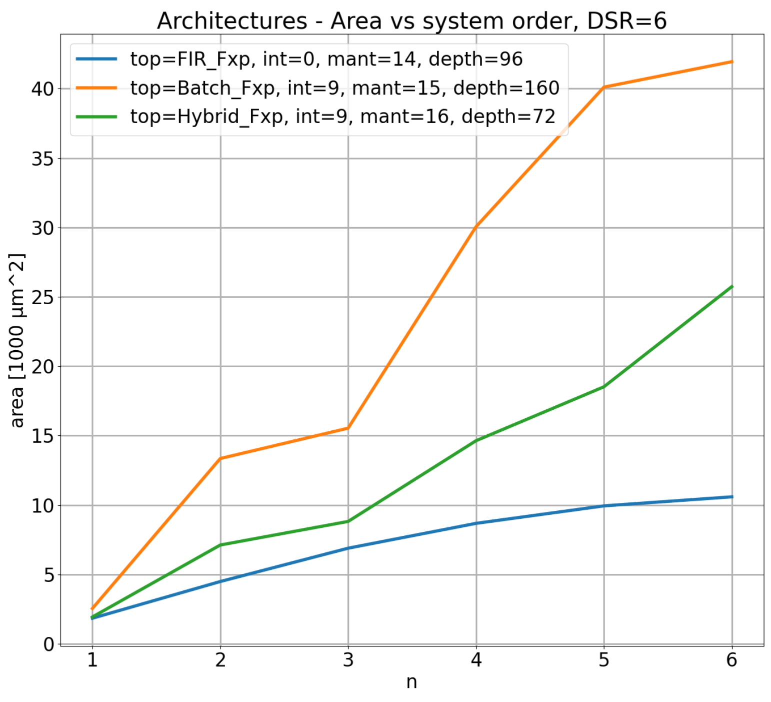


Figure F.1: Basic architectures - Area vs System order, DSR=6

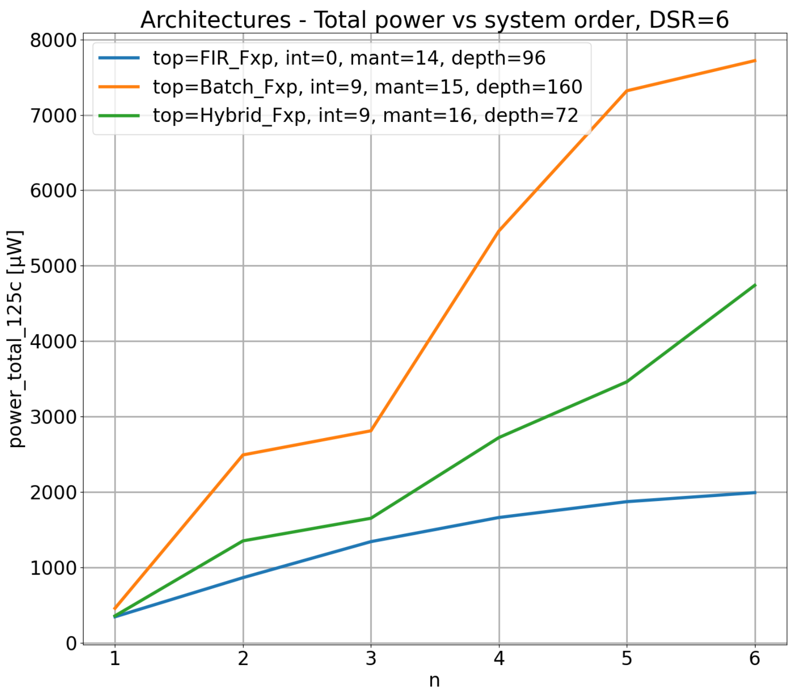


Figure F.2: Basic architectures - Total power vs System order, DSR=6

G DSR area plot

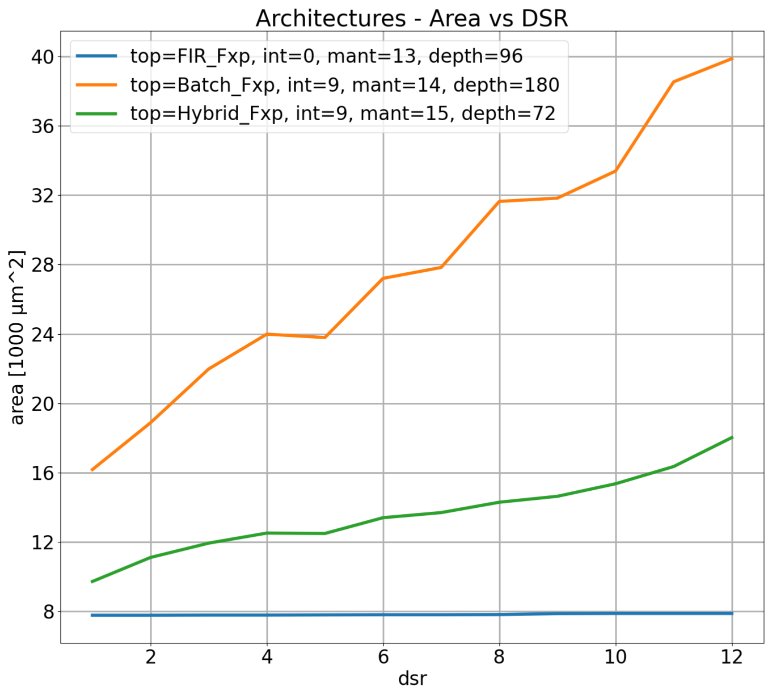


Figure G.1: Basic architectures - Area vs DSR

H Frequency plots

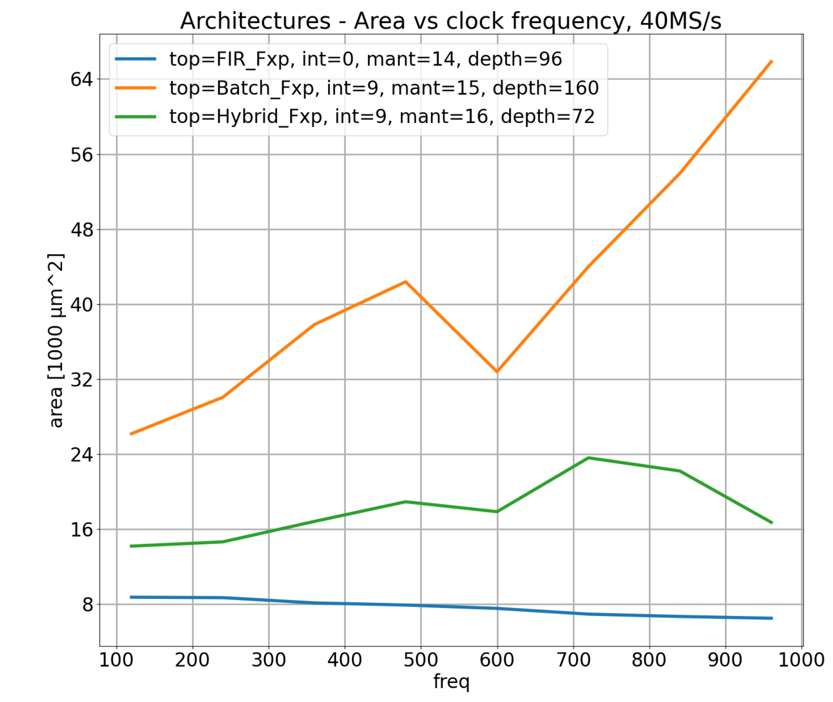


Figure H.1: Basic architectures - Area vs Clock frequency

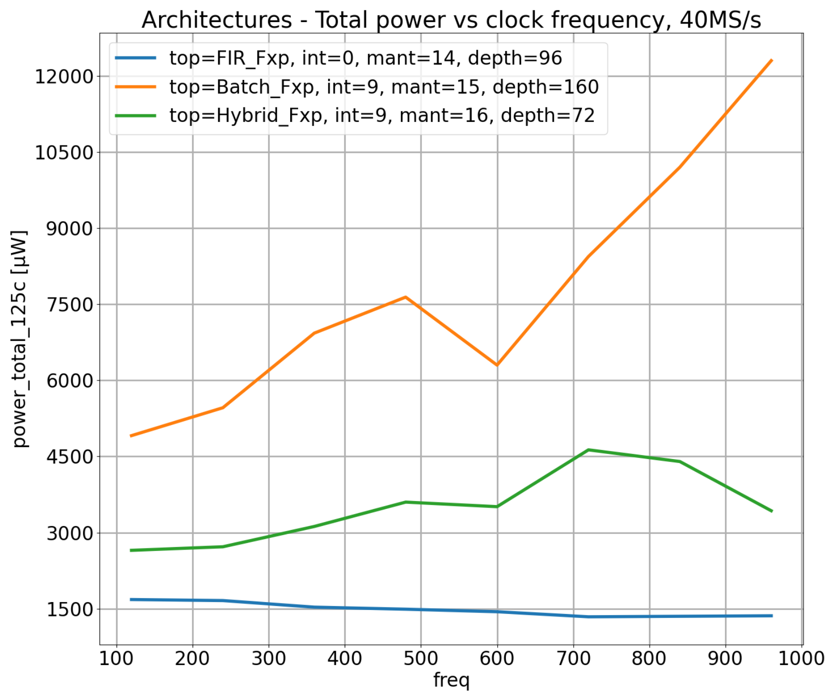
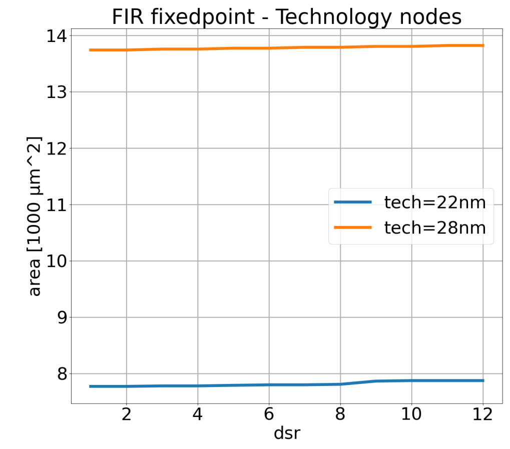
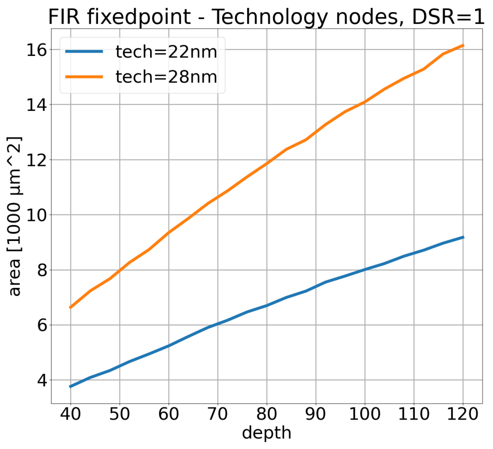


Figure H.2: Basic architectures - Total power vs Clock frequency

I Technology node plots

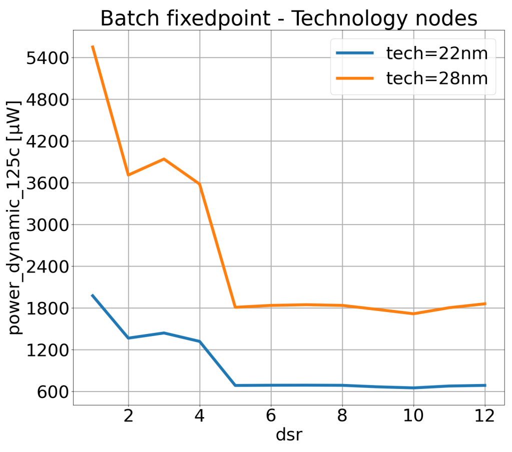


(a) Area vs DSR

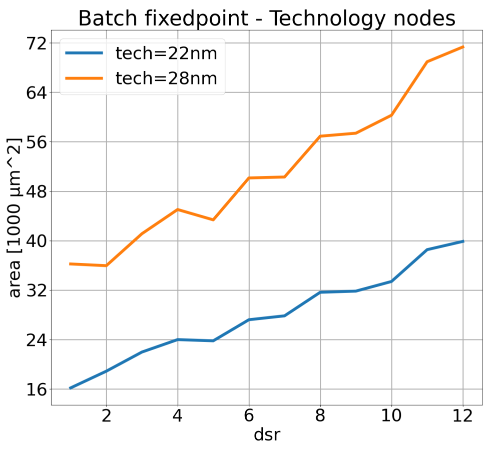


(b) Area vs Filter Depth

Figure I.1: FIR fixed-point - Comparing the 22nm and 28nm technology nodes

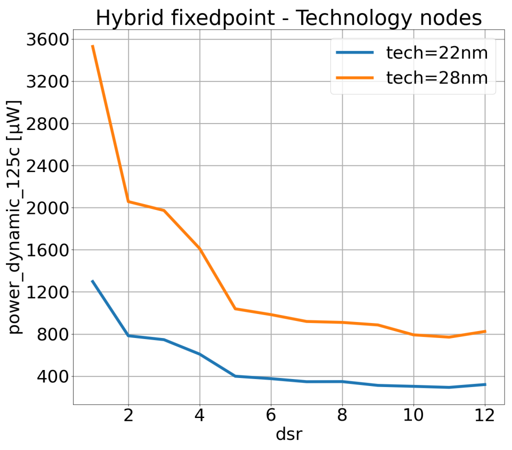


(a) Dynamic power vs DSR

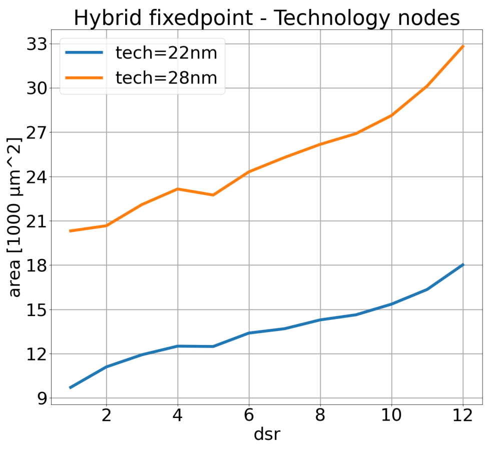


(b) Area vs DSR

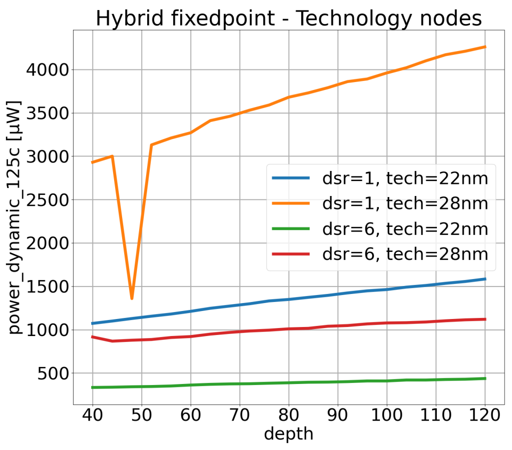
Figure I.2: Batch fixed-point - Comparing the 22nm and 28nm technology nodes



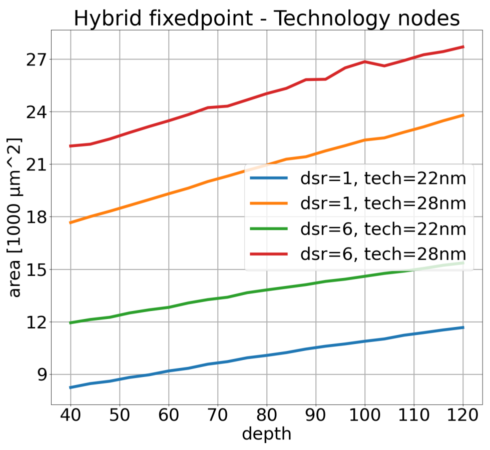
(a) Dynamic power vs DSR



(b) Area vs DSR



(c) Dynamic power vs Filter depth



(d) Area vs Filter depth

Figure I.3: Hybrid fixed-point - Comparing the 22nm and 28nm technology nodes

J Two-stage DSR - area plots

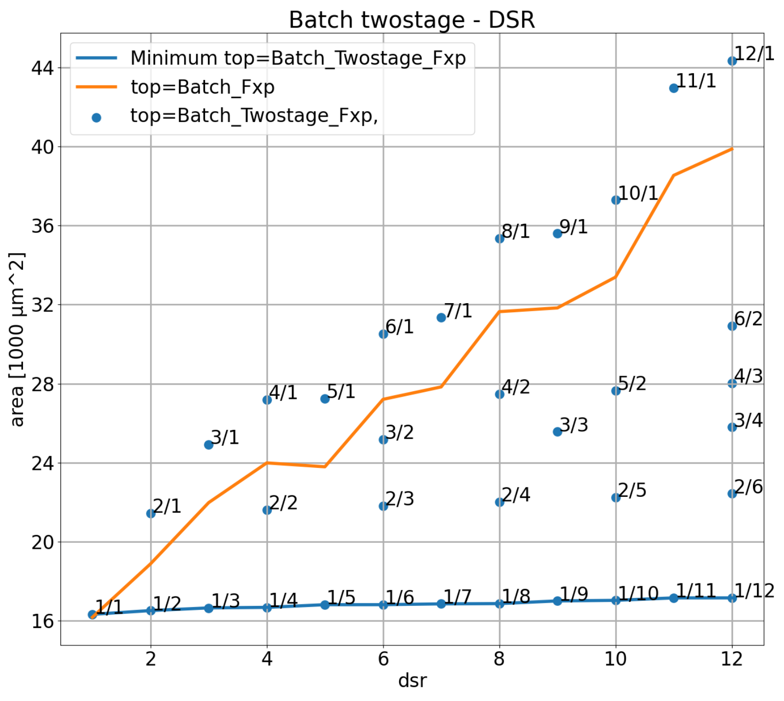


Figure J.1: Batch Two-stage DSR - Area

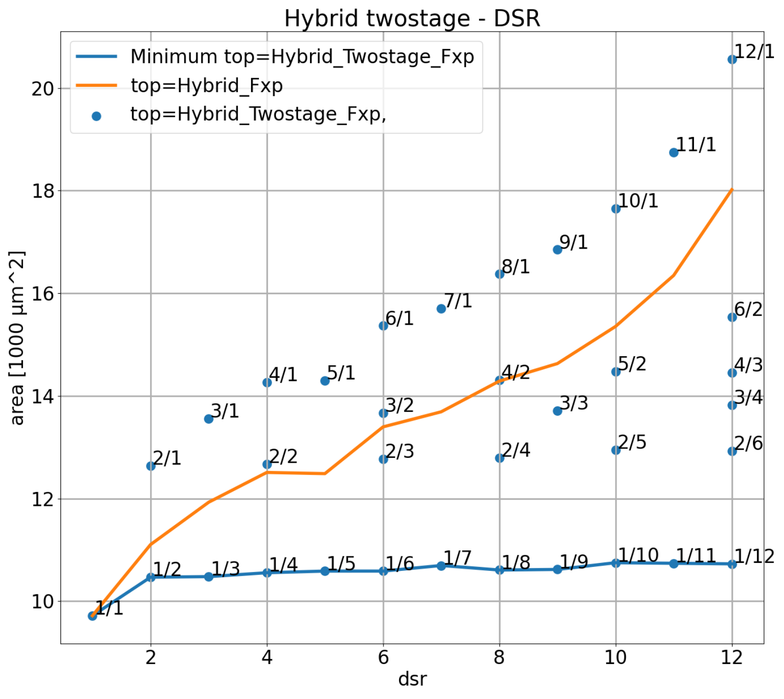


Figure J.2: Hybrid Two-stage DSR - Area

K Scaling vs Mantissa bits

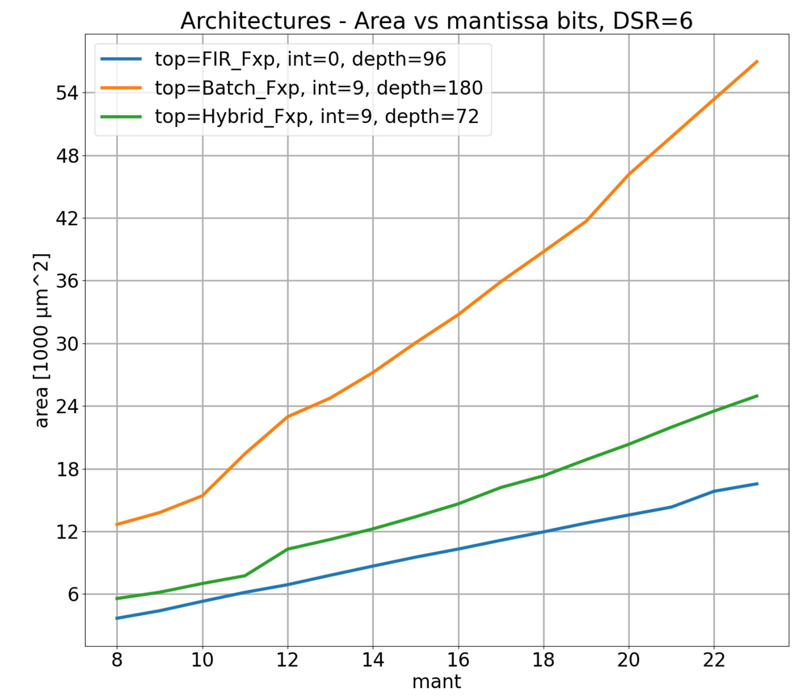


Figure K.1: Basic architectures - Area vs Mantissa bits, DSR=6

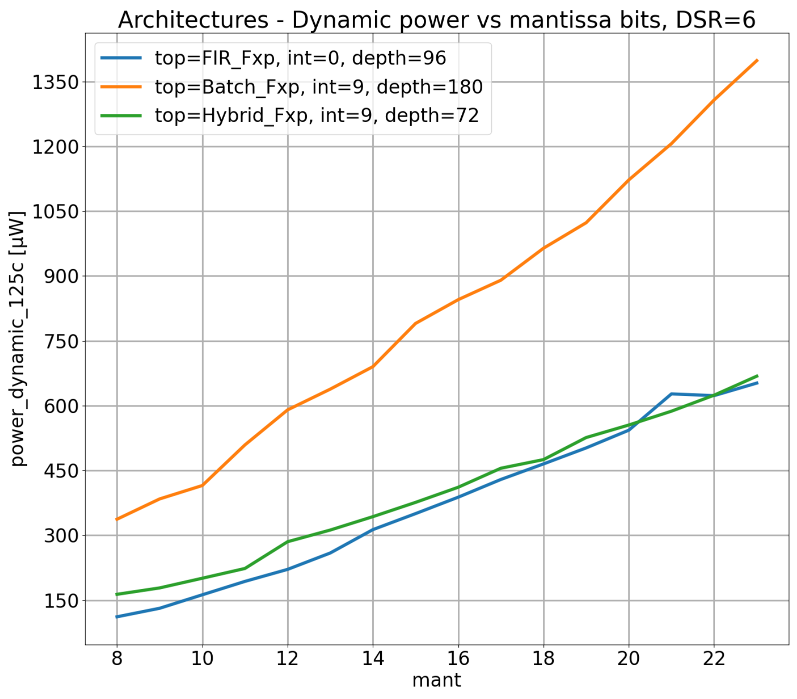


Figure K.2: Basic architectures - Dynamic power vs Mantissa bits, DSR=6

L Cumulative implementation [4]

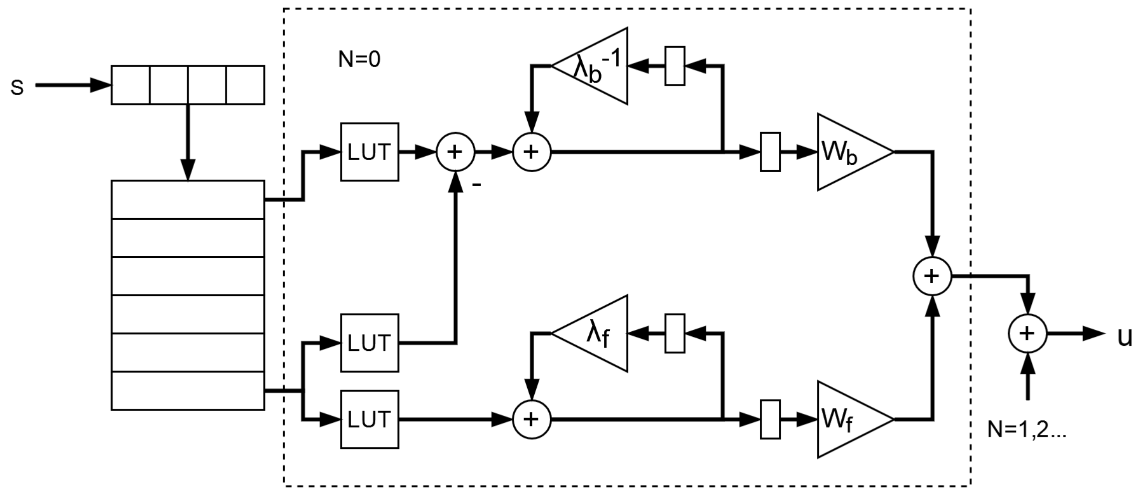


Figure L.1: Cumulative implementation [4]

The cumulative implementation has been of interest as a resource efficient solution, but has not been successfully implemented. Following is the motivation, idea behind it, and problems encountered.

The lookahead of the estimator can be viewed as a gliding window of samples, where one sample is added and another removed at each step. A cumulative approach is often preferred when gliding windows are needed in digital calculations, since it is more resource efficient than summing every element at each step. The challenge of applying a cumulative approach to this estimator is the factor $\overleftarrow{\tilde{\lambda}}_n$ inside the loop.

Equation 2.5 shows the lookahead mean as an infinite recursive calculation, but in practice this range will be finite. Thus it can be rewritten for a lookahead with length L as follows in equation L.1. To make the direction of this recursion the exponent of the loop-factor would need to be negative. Thus it is multiplied with a factor $\overleftarrow{\tilde{\lambda}}_n^{L-L}$ so we get equation L.2.

$$\overleftarrow{\tilde{m}}_n = \sum_{k=0}^{L} \overleftarrow{\tilde{\lambda}}_n^k * \overleftarrow{\tilde{f}}_n(s[k]) \tag{L.1}$$

$$\overleftarrow{\tilde{m}}_n = \overleftarrow{\tilde{\lambda}}_n^L * \sum_{k=0}^{L} \overleftarrow{\tilde{\lambda}}_n^{k-L} * \overleftarrow{\tilde{f}}_n(s[k]) \tag{L.2}$$

So when a sample is added to the recursion its contribution must be multiplied with the factor $\overleftarrow{\tilde{\lambda}}_n^L$ as shown in L.3. Likewise, the sample which no longer should be part of the lookahead must be removed, leading to equation L.4.

$$\overleftarrow{\tilde{f}}_n^+ = \overleftarrow{\tilde{f}}_n * \overleftarrow{\tilde{\lambda}}_n^L \tag{L.3}$$

$$\overleftarrow{\tilde{f}}_n^- = \overleftarrow{\tilde{f}}_n * \overleftarrow{\tilde{\lambda}}_n^{-1} \tag{L.4}$$

Finally we arrive back at a recursive formula shown in L.5. Here the scope of the lookahead is limited to L samples, equivalent to an online implementation of the estimation filter.

$$\overleftarrow{\tilde{m}}_{k,n} = \overleftarrow{\tilde{\lambda}}_n^{-1} \overleftarrow{\tilde{m}}_{k-1,n} + \overleftarrow{\tilde{f}}_n^{+}(s[k+L]) - \overleftarrow{\tilde{f}}_n^{-}(s[k-1]) \tag{L.5}$$

When the direction of the lookahead recursion is reversed its loop-factor becomes larger than 1, which means any errors become magnified. When using floating-point arithmetic this is a problem, since the exponent changing will change the precision of the number. Which makes it really difficult to remove the exact contribution of a sample, making the lookahead recursion unstable. Implementing this with fixed-point arithmetic would solve the issue of the precision changing. However, the data-words would require many bits to give the necessary range. With the analog system used in this project one of the inverted loop-factors was 21, with a lookahead length of 150 the total multiplication applied to a sample would be 21^{150}. More than 660 bits in word-width would be needed to implement this range, which is not feasible.

Maybe this can be implemented with floating-point and some clever way to handle the rounding errors. Maybe it is feasible with some other set of constants. Or possibly with a more exotic number system like logarithmic number representation. But it is not as robust as the other implementations and is currently deemed impractical.