UNIVERSITY OF OULU

FACULTY OF INFORMATION TECHNOLOGY AND ELECTRICAL ENGINEERING
DEGREE PROGRAMME IN ELECTRONICS AND COMMUNICATIONS ENGINEERING

MASTER'S THESIS

Analysis of fixed-point and floating-point arithmetic representations' impact on synthesized area of a digital integrated circuit

Author	Laura Rinne
Supervisor	Jukka Lahti
Second Examiner	Timo Rahkonen
Technical Advisor	Antti Ojala

April 2019

Rinne L. (2019) Analysis of fixed-point and floating-point arithmetic representations' impact on synthesized area of a digital integrated circuit. University of Oulu, Degree Programme in Electronics and Communications Engineering. Master's Thesis, 54 p.

ABSTRACT

This thesis compared fixed-point and floating-point representations, using signal-to-quantization-noise-ratio (SQNR) and synthesized area as key comparison methods. Good-enough SQNR was set to 40 dB, and the goal was to choose area that was as small as possible, but still had sufficient dynamic range (DR), and also fulfilled the SQNR requirement.

Quantization models for both representations were implemented with Matlab. For examination of the SQNR, an algorithm was chosen and aforementioned quantization models were added inside it. The chosen algorithm was memory-based 64-point FFT, implemented with radix-2 butterfly. The performance drop inside algorithm caused by arithmetic representation quantization was examined using SQNR. To be able to calculate the error value, a reference model was implemented, and that was done using FFT-function of Matlab.

When SQNR-analysis had been executed, synthesis was run for arithmetic operation models, for area and power estimate calculation. From these results, a conclusion of impact on area of FXP and FLP on different FFT models was done and a superiority comparison was possible.

Keywords: arithmetic representation, floating-point, fixed-point, quantization, SQNR, IC-area, FFT, dynamic range

Rinne L. (2019) Kiinteän pilkun luvun ja liukuvan pilkun luvun aritmeettisten esitystapojen vaikutusten analysointi digitaalisen mikropiirin syntetisoituun pinta-alaan. Oulun yliopisto, Elektroniikan ja tietoliikennetekniikan tutkinto-ohjelma. Diplomityö, 54 s.

TIIVISTELMÄ

Tässä työssä vertailtiin kiinteän pilkun lukuja ja liukuvan pilkun lukuja, käyttäen tärkeimpinä vertailuparametreina signaalikvantisointikohinasuhdetta (SQNR) sekä synteesistä saatavaa pinta-alaa. SQNR tavoitearvoksi asetettiin 40 dB ja tavoitteena oli valita mahdollisimman pieni pinta-ala, jolla vielä saavutettiin tarpeeksi suuri dynaaminen alue (DR) ja SQNR tavoite täyttyi.

SQNR:n laskentaan tarvittiin molemmille aritmeettisille esitystavoille kvantisointimallit, jotka tehtiin Matlab-ohjelmalla. Lopulta kvantisointikohinan tarkempaan tarkasteluun valittiin algoritmi, jonka sisälle edellä mainitut kvantisointimallit asetettiin. Valittu algoritmi oli muistipohjainen 64-näytteinen FFT, joka on toteutettu radix-2 perhoslaskennalla. Algoritmin sisällä tapahtuvaa aritmeettisesta esitystavasta johtuvaa suorituskyvyn muutosta tutkittiin SQNR:n avulla. Jotta virhe voitiin laskea, myös referenssimalli täytyi implementoida, ja siihen käytettiin Matlabin valmista FFT-funktiota.

Kun SQNR-analyysi oli suoritettu, ajettiin aritmeettisille operaatiomalleille synteesit, joista voitiin laskea algoritmin vaatima pinta-ala. Näistä tuloksista voitiin yhteenvetää liukuvan pilkun ja kiinteän pilkun lukujen vaikutukset FFT mallien pinta-aloihin, ja siten tehdä paremmuusvertailua niiden välillä.

Avainsanat: aritmeettinen esitystapa, liukuvan pilkun luku, kiinteän pilkun luku, kvantisointi, SQNR, IC:n pinta-ala, FFT, dynaaminen alue

TABLE OF CONTENTS

FOREWORD

This thesis was carried out at MediaTek in Oulu as a part of my Master's degree. First, I would like to thank my coworkers at MediaTek for providing an inspiring and supportive atmosphere. Special thanks go to my technical advisor and manager, M. Sc. (Tech) Antti Ojala, for helping me through the process with expertise and patience. In addition, I thank B. Sc. (Tech) Ari Hatula and B. Sc. (Tech) Mika Lehtonen for offering their help and expertise regarding the subject. I thank Chief Engineer Jukka Lahti for supervising this thesis and Professor Timo Rahkonen for second examination of this thesis.

I want to express my gratitude for my fellow students for providing great company and making many memories during all these years. I also want to thank my family for their undying support. Special thanks goes to my boyfriend for his support and empathy during the writing process.

Oulu, April 9, 2019

Laura Rinne

ABBREVIATIONS

ADD	Addition
CMUL	Complex Multiplication
DR	Dynamic Range
DFT	Discrete Fourier Transform
DIT	Decimation in Time
EVM	Error Vector Magnitude
FFT	Fast Fourier Transform
FLP	Floating-Point representation
FXP	Fixed-Point representation
IC	Integrated Circuit
KPI	Key Performance Indicator
LSB	Least Significant Bit
MSB	Most Significant Bit
MUL	Multiplication
OFDM	Orthogonal Frequency Division Multiplexing
RTL	Register Transfer Level
SNR	Signal to Noise Ratio
SQNR	Signal to Quantization Noise Ratio
WL	Word Length
XOR	Exclusive Or

A_{max}	Maximum magnitude of format
A_{min}	Minimum magnitude of format
e_{max}	Maximum exponent
e_{min}	Minimum exponent
FWL	Fractional word length
IWL	Integer word length
m_{max}	Maximum mantissa
m_{max}	Minimum mantissa
N	Number of points in FFT
P_{sig}	Power of signal
P_{err}	Power of error
R	Radix
t	Execution time

1. INTRODUCTION

In digital systems, signals need to be converted from continuous to discrete values. The signal values are represented in discrete binary format, and those representation types are fixed-point arithmetic representation (FXP) and floating-point arithmetic representation (FLP). Both of these representation types have their own features when it comes to quantization. This thesis focuses on creating some guidelines for choosing between representations in different applications.

Some of the most important features in hardware design are area, power and capability to fulfill the performance requirement set by algorithm. In general, those performance measurement tools are referred to as key performance indicators (KPI) and those are separately defined depending on the context and for IC industry, those consist on the key features that make the product appealing to customers.

Quantization is a necessary step in getting a signal to digital domain, because of the limited number of bits available for representing a value. Quantization always introduces error to system, because accurately representing all values in digital domain is impossible. This error caused by quantization is measured and analyzed using signal to quantization noise ratio (SQNR). [1]

As first step of this thesis, custom quantization functions were needed for fixed-point and floating-point representations. Then that quantization was applied into an algorithm, to see how different representations and quantization affect on the performance of an algorithm. A rather simple 64-point radix-2 memory based fast Fourier transform (FFT) algorithm was chosen for further analysis. Anyway, the focus of the thesis was not on the chosen algorithm, but the quantization effects and the areas of arithmetic operations used in the algorithm, in this case adders and multipliers. Memory and registers were also included in area estimation.

To be able to calculate SQNR, a reference value was needed. As most of this thesis work was done with Matlab, it was reasonable to use the default data format of Matlab as reference. That data format is double precision floating point, which has 64 bits, and a wide dynamic range. It was precise enough to get accurate error results.

After analyzing the quantization effects and therefore SQNR values of different representations with different word lengths (WL) and dynamic ranges (DR), it was time to move on to area and power analysis. Area and power results were obtained from synthesis results. Synthesis was run for those arithmetic operations used in FFT algorithm. When areas of separate operations were known, arithmetic area was calculated. Due to the memory-based nature of chosen FFT algorithm, also the required memory area needed to be estimated.

After these steps, it was possible to make comparison between representations and their performance in algorithm case. The key comparison point was nevertheless the area, so the SQNR was used mostly to compare the performance of representations with similar area or dynamic range.

Some needed background theory about the arithmetic representations and their features are represented in chapter 2. Quantization and its effects are explained in theory part, and the most important quantization related topic being SQNR. Some information about FFT algorithm is also provided, so the usage of arithmetic operations inside the algorithm becomes clearer and therefore further explains the area results to be obtained later. The key performance indicators are also examined keeping in mind the needs of IC industry. It is important to understand the theory behind the thesis subject to understand the chapters to follow.

In the practical part of this thesis, different models were implemented using Matlab for both arithmetic representations, and those models are explained in chapter 3. Quantization models were implemented first, because those models were added to the algorithm to examine the performance drop inside the algorithm caused by the quantization. To be able to calculate SQNR values for FXP and FLP with different word lengths and dynamic ranges, also a reference model was needed. In other words, chapter 3 introduces all implemented Matlab models.

To make the area comparison possible, synthesis was run for custom FXP and FLP addition and multiplication operations. The synthesis setup is introduced in chapter 4. The principles of the operations follow the theoretical rules previously introduced in chapter 2. Chapters 3 and 4 are essentially important for understanding of chapter 5, which analyses the results.

Result analysis was performed for quantization models and FFT models, and as comparison points SQNR, dynamic range, area and power were used. After all, it comes down to which representation can give good enough performance with the smallest area. This thesis ends in some conclusions and discussion about the subject and some possible ways to continue the work on this subject.

2. THEORY BEHIND FXP AND FLP COMPARISON

This chapter introduces important theory subjects for better understanding of the thesis. First, different representation types get an introduction. Then some arithmetic operations, addition and multiplication, are theoretically compared, when different representations are used to implement them. After that, theory about quantization and an important feature called signal to quantization noise ratio (SQNR) is introduced. Later in this thesis, SQNR values compare the performance of FFT algorithms implemented with both representations. Therefore, some background information about Fast Fourier transform is provided. It is also important to understand the difference between range and dynamic range. The smallest to largest representable values determine range, and the ratio between smallest and largest absolute value determines the dynamic range. [2]

2.1. Fixed-point representation

Fixed-point representation is the simplest numeric representation. It can consist of sign bit, integer bits and fractional bits, as shown in Figure 1. In this thesis, main focus is in such a fixed point format, that has no integer part, because most of the applications use normalized values, that get values in range [-1, 1]. As FXP's name suggests the position of radix-point is fixed in fixed-point arithmetic. That means that the number of bits before and after radix point is fixed. The sign bit is separate, so the format is sign + magnitude. Other ways to represent negative values are one's complement and two's complement. The most common way to represent signed values is two's complement, which allows easier calculation. The range of fractional-only fixed-point values is [-1, 1), when using two's complement. Negative value one is included in the range, but positive not. [1][2]

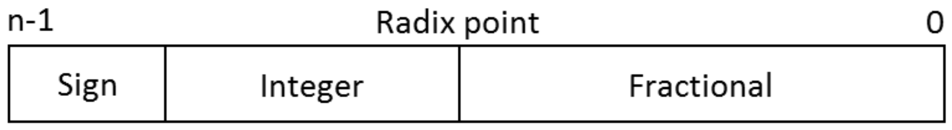


Figure 1. Different parts of fixed-point arithmetic. Zero represents LSB and n-1 represents MSB.

Fixed-point values work like integers that can have the radix-point in any location, not only in the rightmost of the sequence. The bit division of format can be expressed in (U/S)X.Y format, in which the U/S part shows whether the format is signed (S) or unsigned (U), X-part shows the number of integer bits and Y shows the number of fractional bits. In case of unsigned value, the total word length (WL) is the sum of integer bits and fractional bits and in case of signed value, the WL is the sum of integer and fractional bits plus one sign bit. [6]

Below is an example of fractional binary conversion, in which decimal value 0.2 is converted to S0.8 fixed-point format, which has eight fractional bits and a sign bit. Therefore, its word length is nine bits. The zero on the left side of radix point is the sign bit.

$$0.2_{10} = 0.00110011_2$$

The binary sequence of decimal value 0.2 being an endless sequence always introduces error due to truncation or rounding of bits, when converting it to finite word length. When converting the above binary sequence back to decimal, calculation of quantization error shows the amount of error that the truncation caused.

$$0.00110011_2 = 0.19921875$$
$$error = 0.2 - 0.19921875 = 0.00078125$$

Fixed-point arithmetic representation has the same maximum quantization error value throughout its range, when ignoring overflow. This error value depends on the chosen rounding method. In round to nearest rounding method, the maximum error value is half of the least significant bit, and in truncation quantization mode, it can be up to one LSB.

The word length and bit division between parts determine the range and dynamic range of the format. In case of signed magnitude, the maximum binary sequence is all ones for the whole word length after sign bit, and therefore that equals to below equation.

$$|A_{max}| = 2^{IWL} - 2^{-FWL} \approx 2^{IWL} \tag{1}$$

In the equation 1 above A_{max} is maximum value, *IWL* stands for integer word length and *FWL* stands for fractional word length. The value of FWL is subtracted, because in case of sign + magnitude, the maximum absolute value is a bit string of all ones after sign bit, and value 2^{IWL} cannot be exactly represented. The effect of 2^{-FWL} is nevertheless negligible and therefore the maximum value leads to the result being approximately 2^{IWL}. That would be the precise result when two's complement is used. FXP format has maximum absolute value shown in the equation above, and the minimum absolute value shown in the equation below.

$$|A_{min}| = 2^{-FWL} \tag{2}$$

In equation 2, A_{min} is the minimum value and *FWL* stands for fractional word length as previously mentioned. Dynamic range calculation uses previously introduced minimum and maximum absolute values, by dividing the maximum absolute value with the minimum absolute value and then converting it to decibels using logarithm.

$$DR = 20\log_{10}\frac{A_{max}}{A_{min}} \tag{3}$$

The above equation 3 shows the dynamic range calculation, in which *DR* [dB] stands for dynamic range, A_{max} for maximum value and A_{min} for minimum value as previously declared. This form of the equation is valid for both arithmetic representations. [2]

2.2. Floating-point representation

Floating-point arithmetic representation is more complicated than fixed-point representation. It consists of exponent and mantissa, which both have their own sign bit, and as its name suggests, the location of radix point floats, and it is determined by exponent value. Mantissa is usually in sign + magnitude format and exponent is in two's complement format. Therefore, the sign bit of mantissa can be seen separately in Figure 2. In custom FLP-formats, the mantissa can also be represented in two's complement format to make some calculations easier. It is also possible, that the exponent gets positive values only, and negative values are achieved by using exponent bias. For example, that is the case in IEEE 754 standard. [11]

The mantissa of floating point value determines the precision and accuracy and the exponent mostly determines the dynamic range. Mantissa normalization shapes the mantissa to start with binary sequence 0.1. Gradual underflow introduces an exception to that normalization, and next subchapter explains its working principle. [2][3]

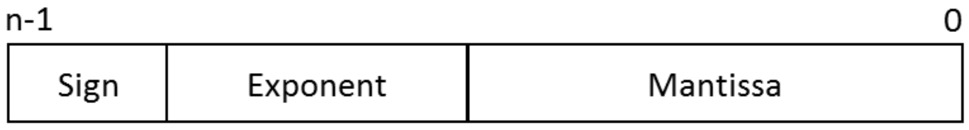


Figure 2. Different parts of floating point arithmetic. Number-zero represents LSB and n-1 represents MSB.

Equation 4 below shows the floating-point representation value calculation. Because binary representation is used, the base is two and mantissa is in binary, in chosen word length.

$$value = mantissa * base^{exponent} \tag{4}$$

As an example, decimal value 0.2 conversion to FLP format goes as shown below. The chosen FLP format has sign bit of mantissa, five mantissa bits and three exponent bits, which means that the word length is also 9 bits, as in previously introduced FXP example.

$$0.2_{10} = 0.11001 * 2^{-2}$$

This also gives some quantization error value, due to truncation of bits, as is shown below. Result shows that with the same word length, the quantization error is bigger in case of FLP, and that is due to the worse accuracy of FLP, caused by the smaller number of mantissa bits compared to FXP bits.

$$0.11001 * 2^{-2} = 0.1953125$$
$$error = 0.2 - 0.1953125 = 0.0046875$$

Considering the nature of floating point values, as was shown in equation 4, the maximum value consists of maximum exponent and maximum mantissa.

$$|A_{max}| = m_{max} * 2^{e_{max}} \approx 2^{e_{max}} \tag{5}$$

In the equation 5 above A_{max} stands for maximum value, m_{max} stands for maximum mantissa value, which is of format 0.1..1 and e_{max} stands for maximum exponent value. The approximation of maximum mantissa is value of one in the equation. For calculation of dynamic range, it is also necessary to calculate the minimum value, as below. [3]

$$|A_{min}| = m_{min} * 2^{e_{min}} = 2^{e_{min}-1} \tag{6}$$

In the equation 6 above, A_{min} stands for minimum value, m_{min} stands for minimum mantissa and e_{min} stands for minimum exponent. The minimum non zero mantissa without gradual underflow is 0.10..0 which is 2^{-1} and therefore the equation can be reduced to final form (6). The dynamic range of FLP comes from the same equation (3) as in FXP. [1][2][3]

2.2.1. Gradual underflow

Without gradual underflow implementation, the underflow occurs suddenly and the value is flushed to zero, when input value normalizes to minimum mantissa 0.10…0 and the exponent is at its minimum. The gradual underflow allows the mantissa to work like FXP in case of underflow, and that means that the mantissa is not normalized to start with its usual binary sequence.

Figure 3 illustrates the practical effect of gradual underflow on quantization levels. It can be seen that without gradual underflow implementation, there are no quantization steps near zero. With gradual underflow, there are steps because the mantissa bits extend and stay denormalized to increase the dynamic range. Figure 3 shows a close-up of underflow, not full range result. Gradual underflow leads to more quantization steps and therefore bigger dynamic range. To calculate the extended dynamic range, new minimum value needs to be calculated to be able to use the equation (3).

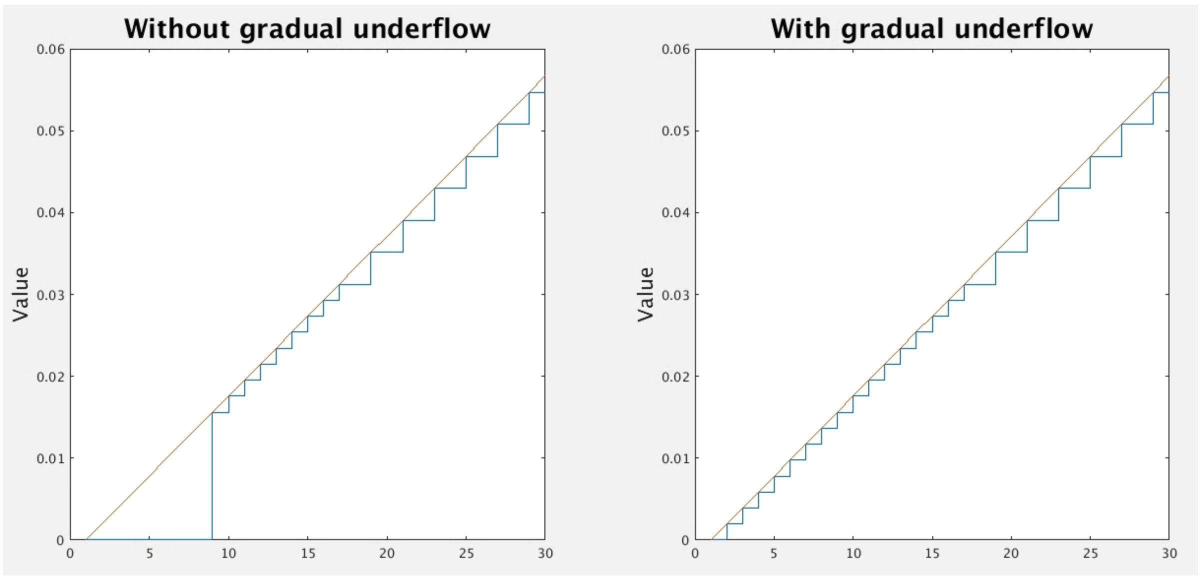


Figure 3. Illustration of gradual underflow's effect on FLP's representable values.

The following equation is used to calculate the minimum value of FLP presentation with gradual underflow implemented.

$$|A_{min}| = m_{min} * 2^{e_{min}} = 2^{e_{min}-MWL} \tag{7}$$

In the equation 7, A_{min} stands for minimum value, m_{min} stands for minimum mantissa, e_{min} stands for minimum exponent and *MWL* stands for mantissa word length. The minimum value of mantissa with gradual underflow has only one '1' in it and it is in the rightmost location (LSB location,) which means that the minimum mantissa value is then $2^{-\text{mantissa\_bits}}$ instead of 0.1 in binary. [4][10]

2.3. Arithmetic operation comparison

Arithmetic operations have some features that depend on the used numerical representation. These typical features affect the area and power consumption of final product. Therefore, it is useful to go through these features for better understanding of synthesis results. Both of the representations have pros and cons when it comes to different arithmetic operations. This subchapter provides information about the most important arithmetic operations for this thesis: addition and multiplication. Different arithmetic representations require different implementations of said operations. One implementation of complex multiplication is using real components, and that approach gets a closer look.

2.3.1. Addition

Because of the integer like nature of the fixed-point representation, addition operation (ADD) is easy to implement. Figure 4 shows the block diagram of fixed-point adder that consists of radix-point alignment and integer-like handling. Radix-point alignment only takes place, when inputs are not in same format, because the format already has fixed the position of radix-point to same location.

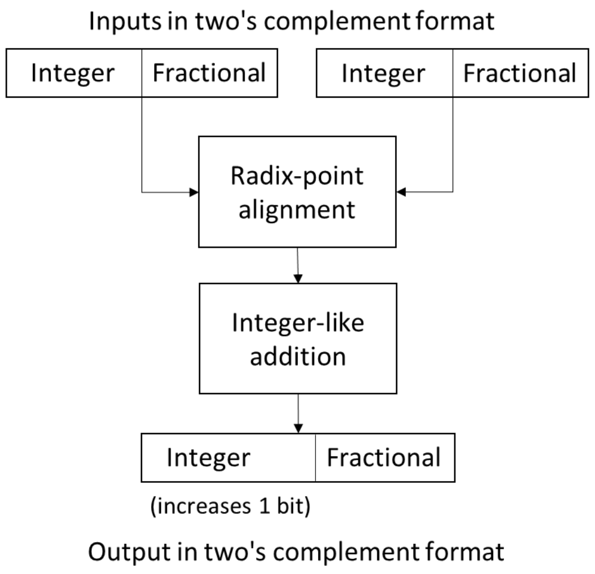


Figure 4. Block diagram of signed fixed-point adder (drawn based on [2]).

Fixed-point addition is easy to implement using two's complement format. Then sign bit does not need to be handled separately, which would be the case when sign + magnitude is used. Addition operation increases the word length of integer bits by one bit. That bit prevents overflow from happening. [2]

In case of FLP adder, the structure of the operation is a bit more complicated. Due to the floating nature of the radix point, exponent comparison is always mandatory in addition of mantissas. In case of unequal exponents, difference in exponent determines the mantissa shifting operation. The mantissa of the value that has smaller exponent value shifts to right as many bits as the exponent difference is. That equalizes the exponents.

Now the mantissas have the radix point at similar location and that enables the mantissa addition or subtraction. The choice between mantissa subtraction and addition follows sign logic as in FXP. Then the result needs to be normalized to start with binary sequence “0.1”, if possible. The normalization process naturally affects the exponent also.

Figure 5 shows the block diagram of FLP addition with sign + magnitude mantissa. Overflow can also occur in FLP addition, even though it is not as common as in FXP addition due to huge dynamic range of FLP. Adding one exponent bit prevents overflow from happening. The increase in dynamic range when adding one exponent bit is excessively big, but necessary to avoid overflow, because adding mantissa bits does not help in avoiding overflow. Full precision result requires adding one mantissa bit also, so none of the result bits are truncated. Nevertheless, normally no bits are added in FLP operations, because of the wide range. In rare case of overflow, the values are saturated. [2]

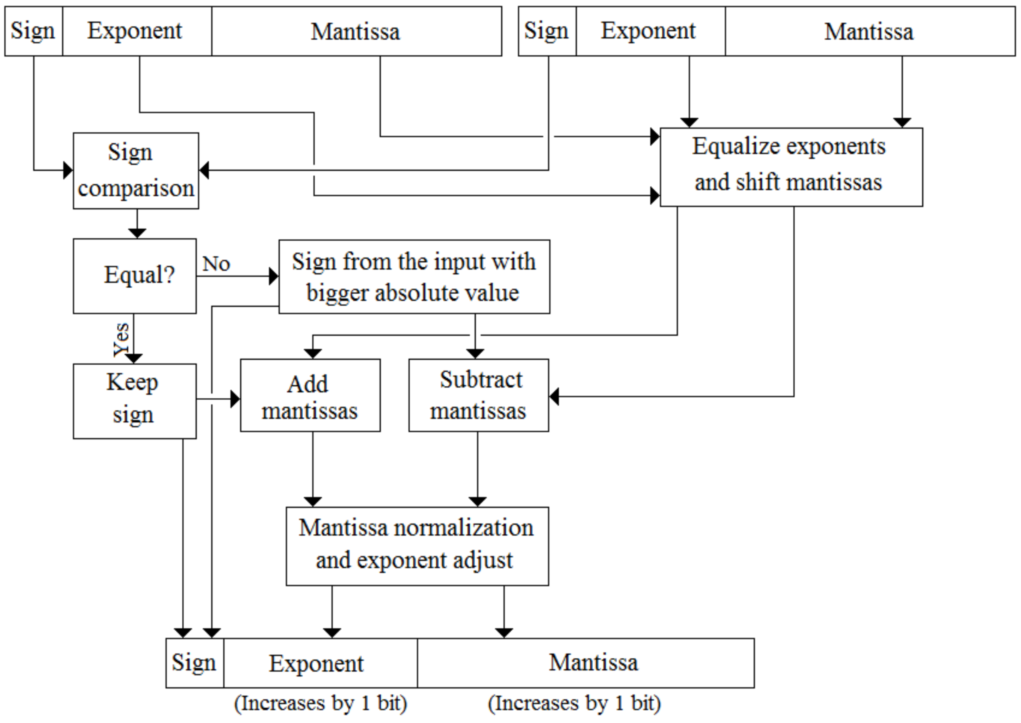


Figure 5. Block diagram of floating-point adder, when mantissa is in sign + magnitude format (drawn based on [2]).

It makes sense that the floating-point adder requires bigger area than fixed-point adder, because of its more complicated structure. Handling mantissa, exponent and in some cases sign of mantissa separately adds the complexity of addition operation compared to integer-like addition of fixed-point.

2.3.2. Multiplication

Another widely used arithmetic operation that gets a closer look is multiplication (MUL). FXP multiplication is more prone to overflow, due to the nature of the multiplication. When multiplying two binary values with each other, the result needs multiple more bits to get the result shown with full precision. The needed word length of the result can reach up to input word lengths added together, which is the case in fixed-point representation. If there are not enough integer bits, overflow occurs. In FLP’s case, overflow does not occur so easily, because the exponent can vary and change the magnitude of the answer. [2]

Figure 6 illustrates the basic working principle of FXP multiplication in sign + magnitude format. Fixed-point multiplier consists of 1-bit multiplications, shifting and partial product additions. In those 1-bit multiplications, other value is multiplied with one bit of the other value, for so long that all of that value's bits have been gone through from LSB to MSB. The result of that 1-bit multiplication is then shifted left. That leads to the first value multiplication with one or zero and therefore the partial results are just zeros and the first value sequence shifted to left. Then those partial products are added together. The sign is handled separately by a XOR operation. This binary multiplication process works similarly compared to multiplication done by hand. The number of fractional bits is the fractional bit amount of multiplied values added together, and that is where radix point is located.

Below is an example of FXP multiplication, and that represents an example of how multiplication affects the word length.

```
     011.01 = 3.25
     010.10 = 2.5
      00000
     01101*
    00000**
   01101***
   010000010 -> 01000.0010 = 8.125
```

If the result was quantized to same format as inputs, the result would be 000.00. In fractional-only calculation, the overflow problem is not present, because multiplication of two fractional-only values cannot produce result that is bigger than one. In case of FXP multiplier with only fractional values, underflow is the problem that occurs when quantizing to same format as inputs. To handle two's complement signed values, there exists multiple different ways to get accurate results in all cases, such as Booth's algorithm or sign extending the multiplier and multiplicand to result WL before calculation. [4]

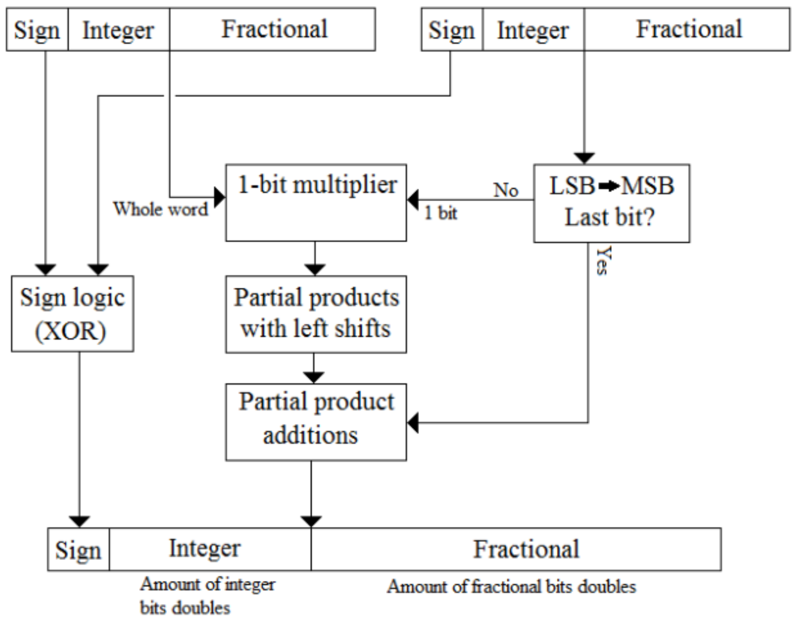


Figure 6. Block diagram of sign + magnitude fixed-point multiplier (drawn based on [2]).

Figure 7 illustrates the FLP multiplication, and it is simpler compared to FXP multiplier, even though exponent and mantissa need to be calculated separately. Fewer operations are needed, because mantissa word length is smaller than fixed-point integer bits and fractional bits added together and therefore there are not as many partial products to be added together in the FXP MUL like mantissa multiplication.

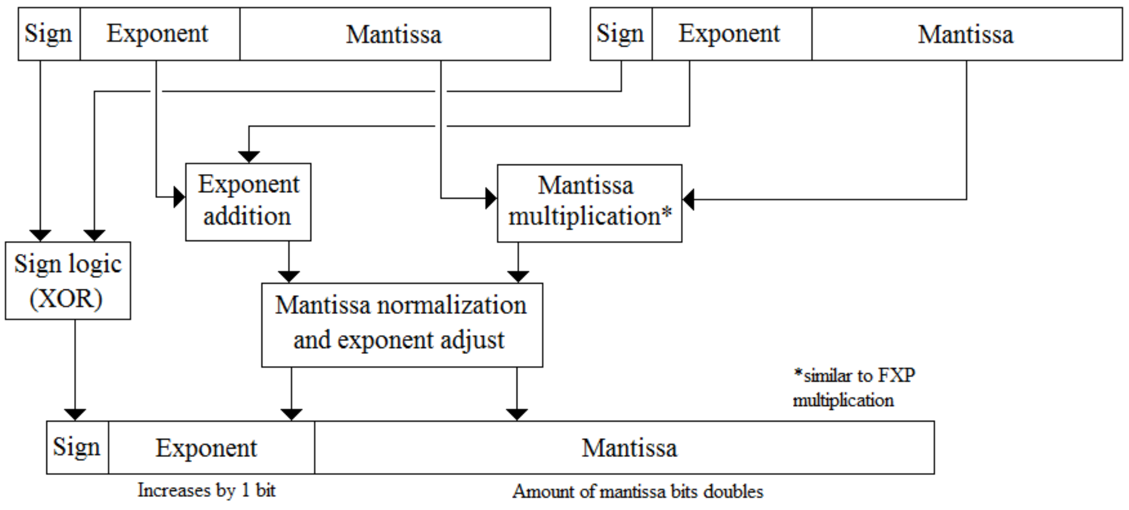


Figure 7. Block diagram of floating point multiplier (drawn based on [2]).

Mantissa handling of FLP resembles the one of FXP operation, when comparing FXP and FLP arithmetic operations. If mantissa is in sign + magnitude format, the addition operation is done without sign extension. In multiplication operation, sign bit does not affect the calculation of the absolute values. Simple exclusive or (XOR) operation determines the sign bit in case of multiplication operation. As usual, in case of negative value, sign bit is one and in case of positive value, it is zero.

Table 1 shows how the result sign is determined using XOR truth table. Also in case of FLP, it is possible to execute the mantissa calculations in two's complement format. In that case, the block diagram would otherwise stay the same, but there would be no separate sign handling, because the sign would be extended as in FXP multiplication.

Table 1. Sign logic, XOR, truth table.

Sign1	Sign2	Output Sign = Sign1 XOR Sign2
1	1	0
1	0	1
0	1	1
0	0	0

2.3.3. Complex multiplication

The complex multiplication (CMUL) can be implemented using real multiplications and additions. Following example shows general case of complex multiplication.

$$(a + ib)(c + id) = ac + iad + ibc - bd$$
$$= (ac - bd) + i(ad + bc)$$

In the example above a, b, c and d are values that are represented in the wanted format. *A* and *c* are real variables and *b* and *d* are imaginary parts of the values. Fixed-point and floating-point CMUL operation consists of ADD and MUL operations of same arithmetic representation, and Figure 8 illustrates that operation. Because CMUL has more multiplication operations, than addition operations, it is possible that it gives slight advantage to FLP, due to its less area-consuming structure of MUL operation. Complex multiplication takes quite big area due to its structure that consists of multiple arithmetic operations. Those arithmetic operations also increase the number of bits needed to express the result accurately.

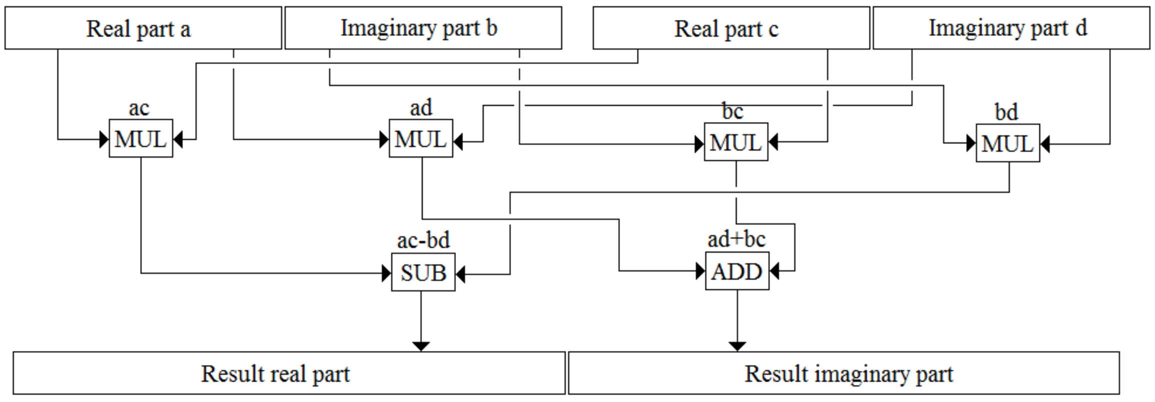


Figure 8. Block diagram of complex multiplication, in which all arithmetic operations are implemented with used arithmetic representation.

Figure 8 represents all real parts and imaginary parts in the chosen format, and to express the result in full precision set by input word length, the output word length extends accordingly. In case of fixed-point representation, the multiplication doubles the number of integer bits and fractional bits and the addition operation adds one more integer bit, and therefore the output word length is 2N+1, when N is the input word length. In case of floating-point representation, number of mantissa bits doubles due to multiplication operation, and the number of exponent bits increases by one. There is no need to add two exponent bits even though both of the operations require extension of exponent range separately, because adding one exponent bit increases the dynamic range enough. [2]

2.4. Quantization

Quantization means the operation that changes the format of value from continuous domain to discrete domain or from digital format to another. Using either of the numeric representations requires quantization and that causes quantization error, in other words quantization noise. Quantization causes quantization error even in ideal cases. The absolute quantization error depends on quantization mode. The difference between FXP and FLP quantization is in their quantization steps.

In FXP quantization, all quantization steps are equal in size: the minimal value difference between two binary values is LSB. Figure 9 illustrates the general quantization steps of FXP. The used word length is one integer bit and two fractional bits, to keep the illustration simple.

It can be seen in the illustration, that the quantization step size is 2^{-2}=0.25 and that all the steps are equal in size. [1]

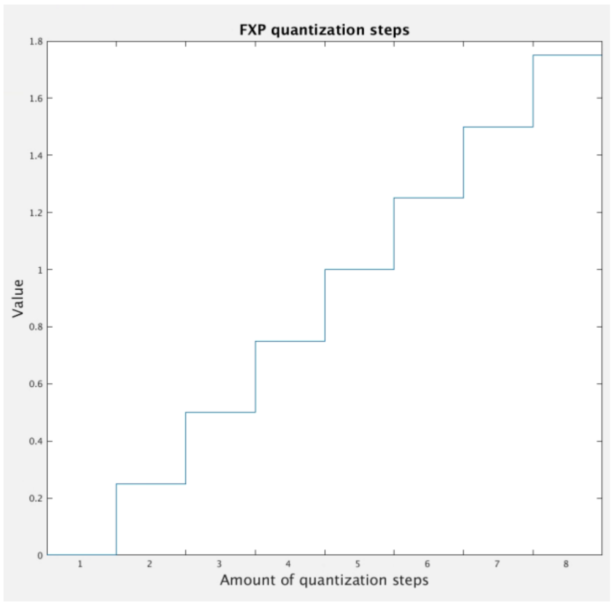


Figure 9. Illustration of representable numbers in fixed-point representation.

In FLP's case, the quantization step is not constant. The size of quantization step changes with exponent value and mantissa bits determine how many steps occur during one exponent value. Figure 10 illustrates those quantization steps and showing that quantization step is smaller near zero. The step size doubles every time the exponent value increases. Without gradual underflow, quantization steps do not exist on the left side of the red line in Figure 10.

The values shown in the x-axis show the exponent values. The used mantissa WL is three bits and exponent WL is two bits. Small WLs make the illustration simple, but the general working principle is the same with bigger word lengths.

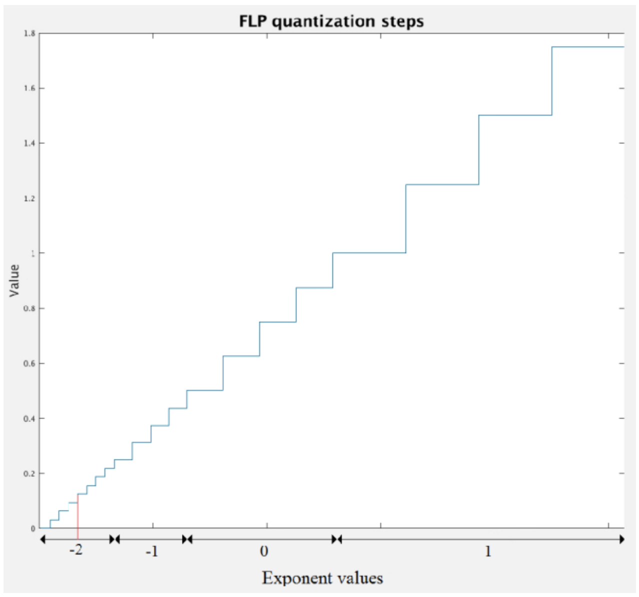


Figure 10. Illustration of representable numbers in floating point representation with gradual underflow.

2.4.1. Quantization error

Quantization error behavior explains the way FXP and FLP quantization work. Sweeping linearly over full range of chosen format, quantization noise behavior comes clear. As expected, FXP has fixed maximum quantization error throughout its range and the maximum quantization error varies in case of FLP. Figure 11 illustrates the differences between FXP and FLP quantization error. That FLP has gradual underflow implemented, and it is visible in the beginning of horizontal axis, where the error has small value. Without gradual underflow, the first error would increase until underflow no longer occurs.

When using truncation as quantization mode, the maximum quantization error value is as big as the current LSB, and therefore it is constant to FXP and changing in FLP. The changes in case of FLP come from the exponent changes when the magnitude of input value increases when sweeping over full range. This means that FLP's error value depends on input value, but relative error is homogeneous and FXP's error value is homogeneous and its relative error depends on input value.

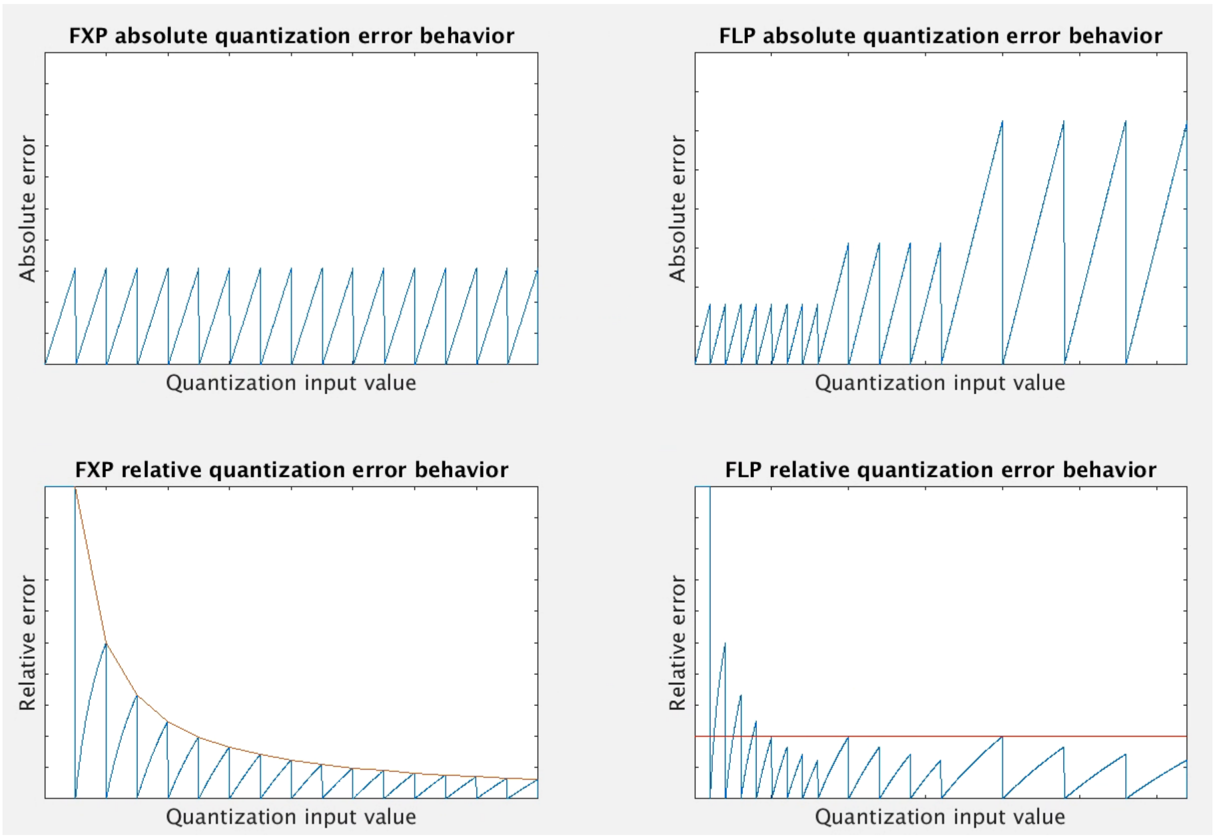


Figure 11. Quantization error behavior of FXP and FLP arithmetic representations.

Those differences are represented in Figure 11, and the red lines in relative error graphs resemble the maximum relative error behavior. It can be seen that in case of FXP, the relative error is very large near zero, but then rapidly decreases. On the other hand, in case of FLP, the maximum relative error stays the same throughout the range. The effect of gradual underflow is also present in relative quantization error of FLP, so that the relative error would rise up to 100 % as soon as underflow occurs. Next subchapter introduces an important way to measure the effect of quantization error, called SQNR. [5]

2.4.2. SQNR

Signal to quantization noise ratio (SQNR) is a quality measure, which measures the relationship between signal level and quantization noise level. SQNR calculation goes similarly to SNR calculation, which takes into account all noise sources. The main difference is that SQNR focuses only on the noise caused by quantization, and that is the point of interest from the thesis' perspective, and the case when considering digital ICs. SQNR is calculated with

$$SQNR = 10 * log_{10} \frac{P_{sig}}{P_{err}} \tag{8}$$

in which *SQNR* [dB] stands for signal to quantization noise ratio, P_{sig} [W] stands for signal power and P_{err} [W] stands for the power of error. The power values P_{sig} and P_{err} are squared average absolute values of error and signal. In this thesis, this is the starting point for the comparison between FFT models. [5]

2.4.3. Saturation

Saturation is one way to deal with overflow that occurs when the value is out of range for chosen format. In other words, there are not enough bits to represent the value. In case of signed values, overflow can be positive or negative. In case of positive overflow, the value saturates to format's maximum and in case of negative overflow, to its minimum. Saturation is illustrated in Figure 12, and it shows a simplified saturation of 3-bit integer to its maximum value seven, when input value is swept from 0-10. That is a case of positive overflow. The blue line represents the continuous value that is quantized using precision of 3 bits. Figure 12 also shows that the used quantization mode is truncation, because the quantized value is never bigger than input value. For example in round to nearest –quantization, the quantized value can be bigger than the input value. [5]

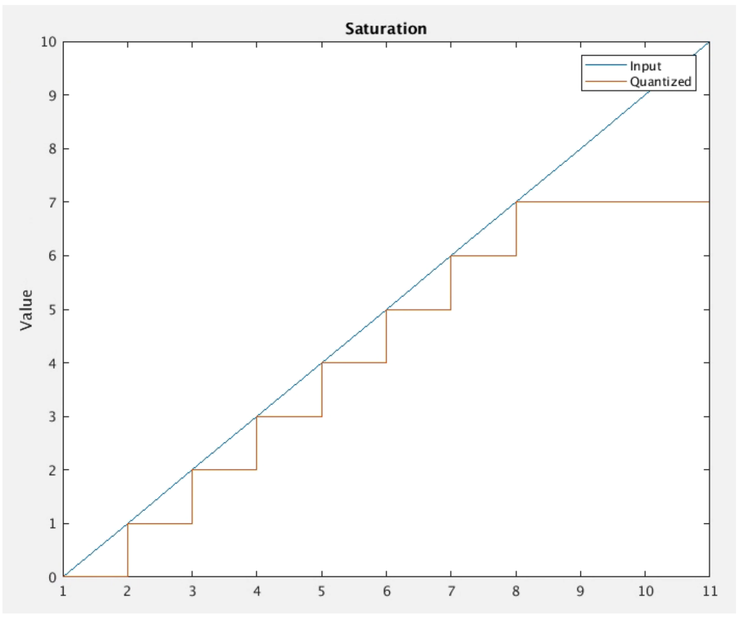


Figure 12. Simple illustration of saturation to maximum value.

2.5. Key performance indicators

Key performance indicators (KPI) measure the performance of a company or a project. The KPIs are separately determined for different applications and companies, keeping in mind the performance, quality and customer friendliness of the goods the company offers. [7]

When considering IC industry and the features that affect the profitability, it is necessary to keep in mind the features that make the product appealing to customer and keep the production expenses as small as possible. KPIs focus on quality and cost, and therefore in IC-industry, main KPIs include area, power and the ability to fulfill performance requirements.

Nowadays, the demand for faster, smaller and overall better products increases and the KPIs need to be set keeping in mind the current project. For example, some projects are destined to design low power applications and some focus more on as small as possible area. The smaller area decreases the manufacturing cost of the chip, and therefore the profit increases, because the customers pay for functionalities, not size, as long as the product can fit in a reasonably sized package. [8]

In case of digital ICs, all noise is caused by the quantization occurring in the circuit. The word lengths in the IC are chosen in a way that the SNR specification is fulfilled. The noise of digital IC consists of quantization noise only, so the SNR can be called SQNR, which only takes into account the quantization noise. Longer word length decreases the quantization error, and therefore makes the SQNR better. [2]

In IC industry, also the speed of the chip belongs to KPIs. An IC with low latency is preferable compared to higher latency. Although, the speed as KPI causes conflicts with other KPIs: low latency chip could cause the area and power dissipation to increase, which is unwanted due to them being KPIs also. [8]

This thesis focuses mainly on area, while power is ignored and performance requirement is set to some chosen value to make comparison of areas more reliable.

2.6. Fast Fourier transform

The algorithm chosen for this thesis shall be FFT (Fast Fourier Transform), because it is commonly used and simple enough for the analysis, but it still includes the most common arithmetic operations: addition and multiplication. FFT is used for example in spectral analysis and linear filtering. It is also a function for OFDM-modems. [9]

Most of the arithmetic operations are additions and the multiplication takes place when twiddle factor comes in. This might give slight advantage to FXP because its simpler adder structure compared to FLP. Although, when looking inside the CMUL operation, the amounts of additions and multiplications increase. All operations in butterfly are complex operations, and that leads to six addition operations and four multiplication operations in the butterfly calculation. The FFT used in this thesis is 64-point radix-2 FFT.

Fast Fourier transform is an algorithm that calculates discrete Fourier transform (DFT) faster and more efficiently than traditional discrete Fourier transform, algorithm. A brief look at DFT calculation is provided in next subchapter. There exists many ways to execute FFT, but in general it samples an input signal over a period-of-time and divides it to its frequency components, basing the calculations in Fourier series. The complexity of traditional DFT is $O(N^2)$ and it can be reduced to $O(N\log N)$ with efficient twiddle factor use. The basic calculation element of FFT is butterfly. Radix-2 butterfly takes two complex points and converts them to new complex points. Figure 13 shows the signal flow graph of radix-2 butterfly. [1]

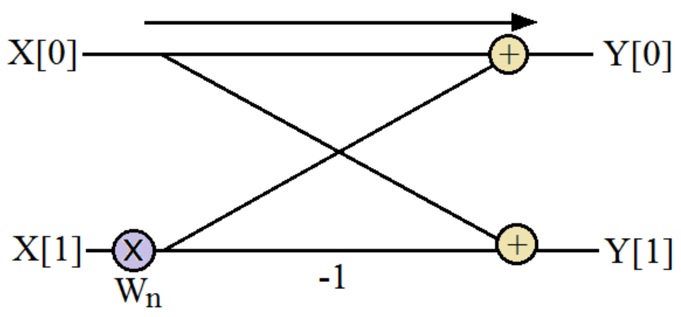


Figure 13. Signal flow graph of radix-2 butterfly that connects the X inputs to Y outputs.

The W$_n$ in Figure 13 represents twiddle factor that is a trigonometric constant coefficient, which enables FFT calculation. The said twiddle factor is calculated with:

$$W_N^{kn} = e^{-\frac{j2\pi}{N}kn} = cos\frac{2\pi kn}{N} - i\ sin\frac{2\pi kn}{N} \tag{9}$$

in which the W_N^{kn} stands for twiddle factor, *N* is number of points of FFT and *k* is from 0 to N-1. This twiddle factor is used to multiply the other input of FFT. The twiddle factor creates the phase rotation to input. [5]

2.6.1. Discrete Fourier transform

The discrete Fourier transform is used for example in spectral analysis. Calculation of DFT is quite slow, so FFT is often implemented. In discrete Fourier transform, a set of complex values is transformed to different set of complex values, and the process is defined by following equation.

$$X_k = \sum_{n=0}^{N-1} x_n e^{-\frac{i2\pi}{N}kn} \tag{10}$$

The above equation (10) reduces to the form below by using Euler's formula, which replaces the exponential part of the equation with sine and cosine.

$$X_k = \sum_{n=0}^{N-1} x_n W_N^{kn} = \sum_{n=0}^{N-1} x_n [\cos\frac{2\pi kn}{N} - i\ sin\frac{2\pi kn}{N}] \tag{11}$$

In equation (11) *X$_k$* is the transformed value, *N* stands for the number-of-points in FFT and *k* is the index of new value, from zero to N-1. [5]

2.6.2. Memory-based FFT

The memory-based, also called iterative, FFT architecture usually has only one butterfly structure implemented on the chip. That one butterfly executes all the calculations iteratively and loads the results of calculations to memory, to correct memory address. The number of iterations needed to compute full FFT is the number of stages in FFT, and that means that the results of current iteration fill up the memory. Overwriting of previous results occurs. Iterative execution takes some time of course, and that execution time is calculated with following equation.

$$t = \frac{N\ log_2 N}{r\ log_2 r} \tag{12}$$

In equation 12, *t* stands for execution time, expressed in clock cycles, *N* stands for the number-of-points in FFT and *r* stands for the used radix. That leads to memory-based architecture to have disadvantage of quite long execution time, and advantage of saving hardware cost by making the area smaller. Bigger radix and higher clock frequency are some ways to shorten the execution time.

The memory-based FFT architecture requires also the memory and address generator, in addition to the butterfly structure. Figure 14 illustrates the simplified working principle of radix-r memory-based FFT. Iterative FFT architecture is unable to calculate FFT, if data continuously arrives at input, because the input first stored to memory requires complete calculation before accepting new input to FFT. The memory block is capable of storing N times two words of chosen word length. Adding separate input buffering for memory-based FFT makes it suitable for processing continuous data. [18]

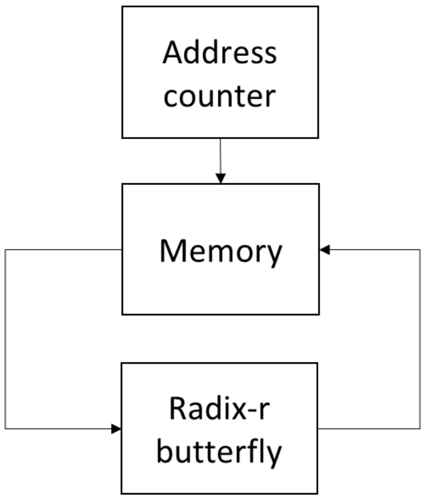


Figure 14. The working principle of memory-based FFT.

2.7. Previous work

Quite a lot of work has been done previously considering the features of floating-point and fixed-point arithmetic representations that mostly focus on either one of the representations, and not that much on the practical differences of them. In those that can be found, there are conflicts regarding the comparison of different numerical representations.

One previous thesis work related to FFT SNR can be found (done in Tampere University), but it focuses only on SNR, and area and power are not considered at all because of the different perspective. It does not quite fulfill the needs considering the choosing of correct numerical representation for hardware design because it completely ignores the area and power consideration. Our goal is not to focus on FFT too much, but rather examine the way arithmetic operations effect on the performance. The previous thesis also focuses much on different rounding methods; while in this study, the focus is in how to fulfill some performance requirement with as small as possible area. In addition, dynamic range is ignored in that previous work. The SQNR calculation is relatively similar to this thesis, because FFT-function of Matlab is used as a reference. [13]

As previously stated, there are many studies considering one of the two arithmetic representations. One done in University of California focuses on fixed-point accuracy on different FFT-algorithms. That gives some great insight on the effect caused by choosing the FFT implementation. This study also uses SQNR as comparison method between FFT-algorithms. [14]

One relatively new study (2017) takes area, energy and delay under consideration and makes comparison between FXP and FLP customized arithmetic in K-means clustering algorithm. This study was interesting because of different algorithm, and that helps in seeing the differences of FXP and FLP when results can also be compared to similar results about for example FFT algorithm, which seems to be one of the most studied algorithms. In this previous study about K-means, FXP holds the advantage especially in case of adders. The advantage is on FLP in case of multiplier but it is not as apparent as in case of FXP's advantage on adders. The energy consumption advantage lays of FXP all the time. [15]

One study takes the previous study a bit further, and compares different algorithms, including for example FFT and K-means. It also compares addition and multiplication operations separately. That "raw comparison" grants the advantage for FXP in all aspects at hand. FLP offers better accuracy/energy tradeoff compared to FXP, but the area advantage still stays with FXP. That leads to FLP being costly compared to FXP. This study concluded that FLP's greatest advantage lays in small WL applications. It also seemed that this previous work focused mostly on power savings. [16]

A big part of comparison between FXP and FLP focuses on audio. Many of those studies also have conflicts and seemingly polarized views on subject. Some studies show that floating-point is easier to program, but the end result may not be as good as wanted, compared to FXP with word length optimization process. The usage of FLP in audio applications can cause the sound to pump. [17]

Previous works mostly agree on the properties collected in Table 2, even though some conflicts exist, mainly on precision and power part.

Table 2. Expected advantages for both arithmetic representations based on previous works on subject.

Property	FXP advantage	FLP advantage
Dynamic Range		X
Precision	X	
ADD area	X	
MUL area		X
Ease of use		X
Power	X	

3. SQNR TEST SETUP

Fixed-point and floating-point representation comparison tool was Matlab. There are no built-in models for quantizing values to wanted word lengths and formats, without using separate toolboxes. That created a need for implementing quantization models. First FXP and FLP got their own quantization models implemented and that allowed studying error values and quantization steps.

Later quantization models were added to custom fixed-point and floating-point fast Fourier transform algorithm models. There were three FFT models: FXP FFT, FLP FFT and reference FFT. The reference model was implemented using built-in FFT function of Matlab. The default format of Matlab is double precision floating-point so all Matlab calculations use said format. The quantization functions just changed the values to represent the values in such precision that matches the other formats. Double precision floating point includes sign bit, 11 exponent bits and 52 mantissa bits. The word length of double precision floating point is 64 bits, and Figure 15 illustrates the bit division between different FLP parts. [11]

Reference FFT model enabled the error and SQNR calculations, when compared with custom FFT models, that had quantization implemented.

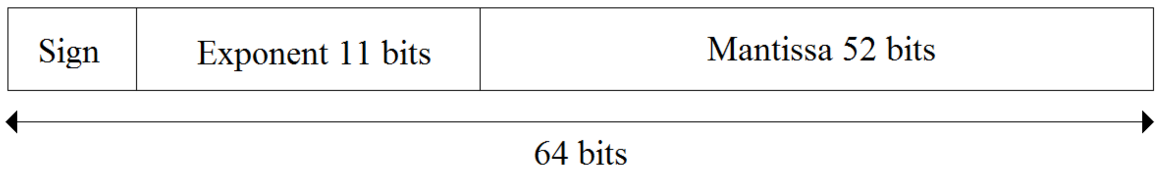


Figure 15. Double precision floating point, which was used as reference for all created Matlab models.

The used fixed-point format consisted of sign bit and fractional bits, and the format was simply referred to as "S0.(fractional bits)". In case of floating-point, the format consisted of mantissa sign bit, exponent bits and mantissa bits. The mantissa was normalized to start with binary sequence 0.1 unless underflow occurs, and gradual underflow took place. The used exponent format used only values between zero and its minimum value.

The reason behind encoding the exponent was to make the ranges relatively equal between FXP and FLP. The encoding prevented one unused bit in FLP, because similar data went to both models, and FXP limited the maximum value because of its fractional nature. Next subchapters explain all different Matlab models in further detail.

3.1. Quantization models

This subchapter introduces the quantization models to get better understanding of the arithmetic representations and the errors they introduced. The same quantization models found their place between FFT algorithm's stages. Looking at single quantization point gave great insight and eased the process and analysis later, because quantization causes most of the error in digital systems.

3.1.1. FXP quantization model

The FXP quantization function took the raw value, desired number of integer bits and desired number of fractional bits as input and then outputted the quantized value. The FXP quantization operation was easy to implement with Matlab, as needed operations are bit shifting and truncation. The bit shifting moved the radix point to rightmost location allowed by the word length. The number of bits on the left side of the radix point was the full word length of the format, including also the fractional part. Then the value was truncated using fix-function of Matlab, and then bit shifting took place so that the radix point location was correct to the format. The bits shifted by multiplying the input with power of two, determined by the number of fractional bits. Figure 16 shows the flowchart of the FXP quantization operation.

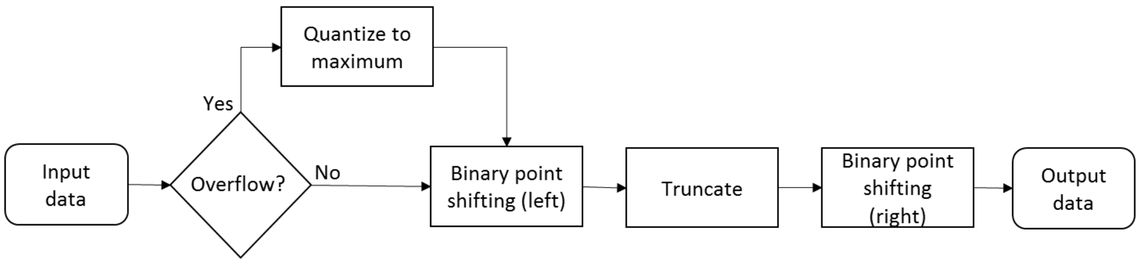


Figure 16. Flow chart of fixed-point quantization Matlab model.

Quantization function also included the saturation of out-of-range values. The quantization function handled complex input values, by dealing with real and imaginary parts separately, following the previously introduced manner.

This quantization function made it possible to see how error behaves with different word lengths and allowed some scripts to be written, for example sweeping the whole range with much smaller step than LSB value, and then calculating error value and relative error. Used reference was double precision floating point, which is the default in Matlab. Also different world lengths were sweeped in the same script, and full range error calculations were done to all wanted word lengths.

3.1.2. FLP quantization model

Two FLP quantization models were created. First, one whose exponents had full range of values, which means positive and negative values and then one with encoded exponent, which only got negative values. This encoding of the exponent was done, because the range of usual data in these applications are within range [-1, 1), so there was no need for the values above that. As previously declared, this led to saving one bit in exponent. All SQNR calculations were done using the model with encoded exponent, and therefore the other model was not visible in test setups.

The quantization function of FLP was a bit trickier to make with Matlab, because exponent and mantissa require separate handling and overflow checking. Saturation to biggest absolute value happened in case of overflow. First step was to find the exponent value by rounding the two base logarithm of input value toward negative infinity. That exponent needed addition of one to normalize mantissa to wanted format. To obtain mantissa value, bit shifting took place, according to the exponent value. Gradual underflow occurred in possible underflow situations. The gradual underflow offered great advantage on small values. Figure 17 is the flowchart of FLP quantization model, and the input data is in double precision floating-point format.

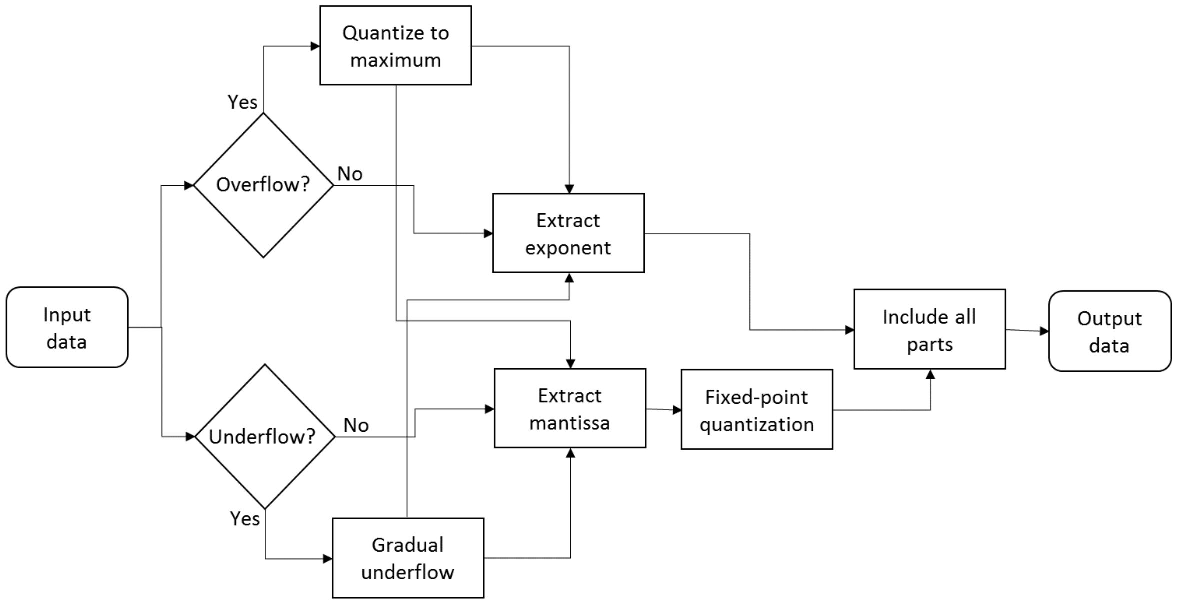


Figure 17. Flowchart of floating point quantization Matlab model.

3.2. FFT models

There were four different FFT models, one for each arithmetic representation and two reference models, in which the difference came from input and output data quantization. Figure 18 shows the way different FFT models were used. Same input data was fed to FXP and FLP FFT models. The chosen algorithm was 64-point radix-2 memory based FFT, so it had six stages, and quantization occurred after all those in FXP and FLP FFT models. The used FFT architecture in FXP and FLP was memory-based FFT, which meant that one butterfly structure did all computations iteratively. Calculating one stage required 32 butterfly calculations, and therefore the whole FFT required 192 butterfly calculations. That was also the amount of clock cycles it took and equation 11 led to that same result about the execution time. Figure 19 explains the idea behind FFT models.

In FXP and FLP FFT models, quantization took place after every stage, in addition to input and output quantization. In other words, all calculations were made with quantized values. This caused quantization error inside the algorithm and therefore made the performance worse. It is the goal of this work to see the tradeoff between performance drop and area. As reference models, two different approaches were used, one with input and output quantization and one without any quantization that uses double precision floating point throughout the calculation. The first reference data was not so prone to performance drop caused by saturation because its final output also saturated when FXP or FLP models' output saturated.

The used FFT model was implemented using decimation in time (DIT), which means that the input was in bit reverse order, and output in correct order. That is why there is input permutation in Figure 19. One other property of the DIT FFT was that the twiddle factor multiplication happened before the butterfly calculation.

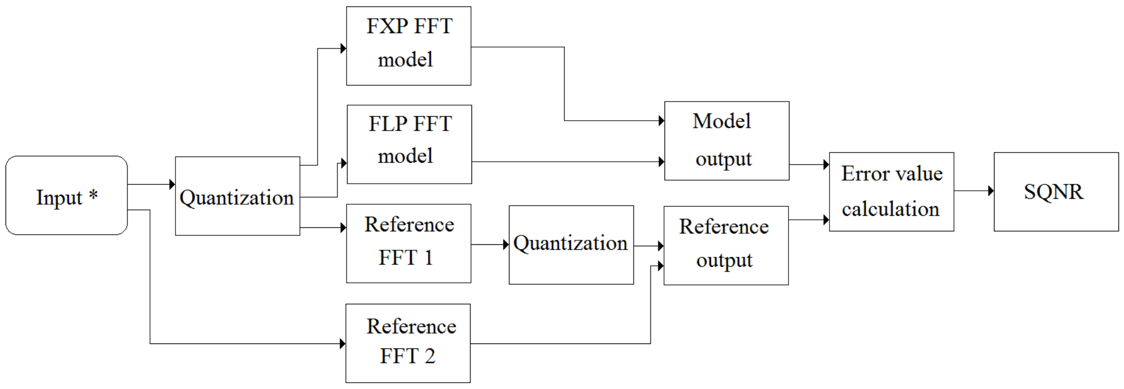


Figure 18. FFT comparison block diagram that shows how the different models were compared.

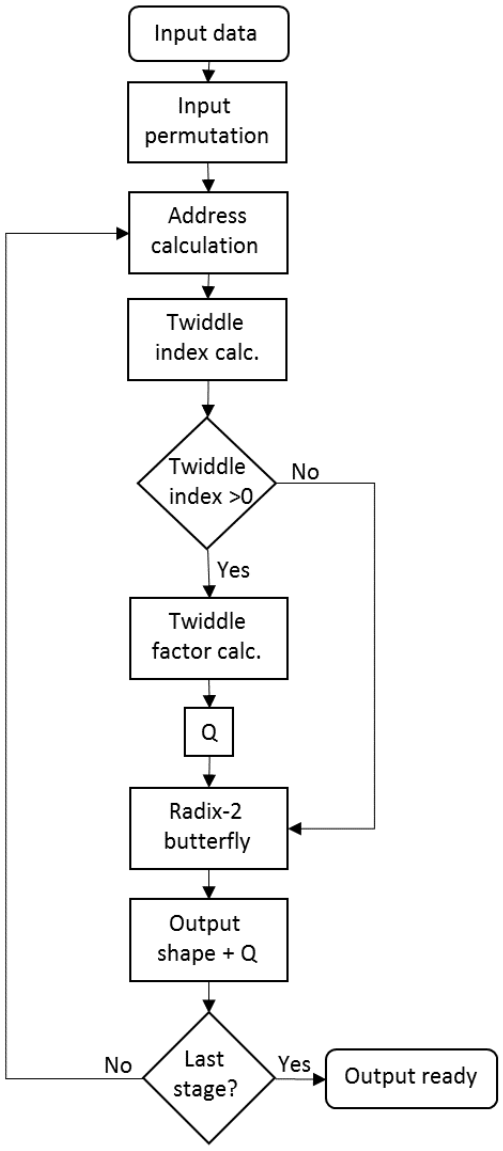


Figure 19. Flowchart of memory-based FFT model, to which the FXP and FLP FFT models were based on.

Table 3 summarizes the properties of FFT part of this thesis. The quantization mode and word length properties were accurate also in case of plain quantization model calculations. Reliable results required multiple reruns with different random input data, because then SQNR calculation could use mean values of error and signal powers. The SQNR requirement was set to 40 dB, which corresponds to 1 % error vector magnitude (EVM), which is commonly used error requirement for baseband. [12]

Table 3. FFT model usage summary.

Specification	FXP	FLP	
Quantization mode	Truncation	Truncation	
Minimum WL (sign included)	9	Exponent	Mantissa
		2	6
Maximum WL (sign included)	24	Exponent	Mantissa
		4	16
SQNR comparison value	40 dB (1 % EVM)	40 dB (1 % EVM)	
Input scaling	9 % of maximum value	9 % of maximum value	
Other notices	Input values have always at least 3 most significant bits as zero	Gradual underflow Encoded exponent	

3.2.1. FXP FFT model

Previously introduced fixed-point quantization model took place in all stages of fast Fourier transform. The same quantization affected the twiddle factors. The quantization occurred inside the algorithm in a way that did not allow the word length to increase inside it.

Matlab-scripts allowed simple sweeping of different word lengths and their SQNR calculations. Matlab plots helped illustrate results and therefore made analysis easier. The plots included also some additional values to make comparison easier.

In FXP FFT model, the used word lengths varied from nine to twenty-four bits. Those word lengths included sign bit, and therefore the number of fractional bits varied between 8 and 23.

The word length did not increase between stages, and all implemented quantization blocks were similar. The Q-letters in Figure 19 represent the quantization operations to current chosen word length of FXP arithmetic representation. Only the first input to radix-2 butterfly needed to be separately quantized, because the output of the butterfly was always quantized before writing it to memory and the following butterfly inputs were previous outputs read from the memory and therefore already quantized to correct format.

3.2.2. FLP FFT model

The working principle of FLP FFT model was the same as the one of FXP FFT model, but the quantization performed between stages was FLP quantization. In FLP FFT model, exponent word length varied between two and four, and it had only negative values and zero, which means that there was no separate sign bit. The number of mantissa bits varied between six and sixteen. Therefore, the whole word length varied between eight and twenty.

3.2.3. FFT reference model

FFT-function of Matlab was the reference for custom FFT models. It executed all calculations inside FFT using double precision. The reference was built with two setups: one with input and output quantization, and one without. The one with input and output quantization implemented, was referred to as Reference FFT 1 in Figure 18. That version was implemented to see the performance drop caused by quantization inside the FFT-algorithm. It made the SQNR tolerate possible saturation to largest value, because reference output also saturated to maximum value in case of overflow. Without input and output quantization, referred to as FFT 2, saturation dropped the SQNR rapidly, and even small amount of saturated values caused the SQNR value to saturate to some value. This version allowed examination of all aspects of arithmetic representations.

3.2.4. Input data

The input data was generated using randn-function of Matlab, so that the generated input signal followed Gaussian distribution and was, in other words, white Gaussian noise. Input values were generated for all real and imaginary parts of 64 points and that was repeated five thousand times, so the FFT could be run through with different input values and then the mean error value, and thereafter SQNR, could be calculated from larger dataset.

Input data was then scaled down to avoid overflow in FFT output. The FFT input needed to be scaled down so that its maximum value was 9 % of maximum value of the current format and that scaled input was fed to FFT models. The input using such a small part of full range causes the input to have several zeros in the beginning, especially in case of FXP and FLP with short exponent.

This could be avoided in some applications by starting with smaller WL and increasing the WL by needed number of bits after each stage. Although this is not a great solution if encoded exponent is used, because it expects the output values to be between ~ -1…1 and encoded exponent ignores the integer part completely. In case of FXP, some integer bits could be added. Fractional bits do not help in case of overflow.

Adding bits in the middle of calculation was also not possible in this case, because the adders and multipliers had some fixed word length, as did the memory. That was due to the iterative nature of the FFT. That meant that the butterfly structure needed to be implemented using the longest required word length, and that alone determines the area.

The input data followed Gaussian distribution to make the comparison more reliable. Gaussian distribution focuses on small, near zero values, which might have given FLP an advantage. Making correct type of input data required generation of 256 random values, because the FFT algorithm was 64-point. Correct IQ-data consisted of taking first generated value as real part, second value as imaginary part and discarding two following values. All 64 points got this treatment, and 128 random values found their place in input IQ-data.

To make the calculations reliable, this whole 64-point input data-generation sequence was repeated five thousand times, forming an input matrix. Figure 20 illustrates the generated real and imaginary inputs, and it can be seen that those histograms follow Gaussian distribution.

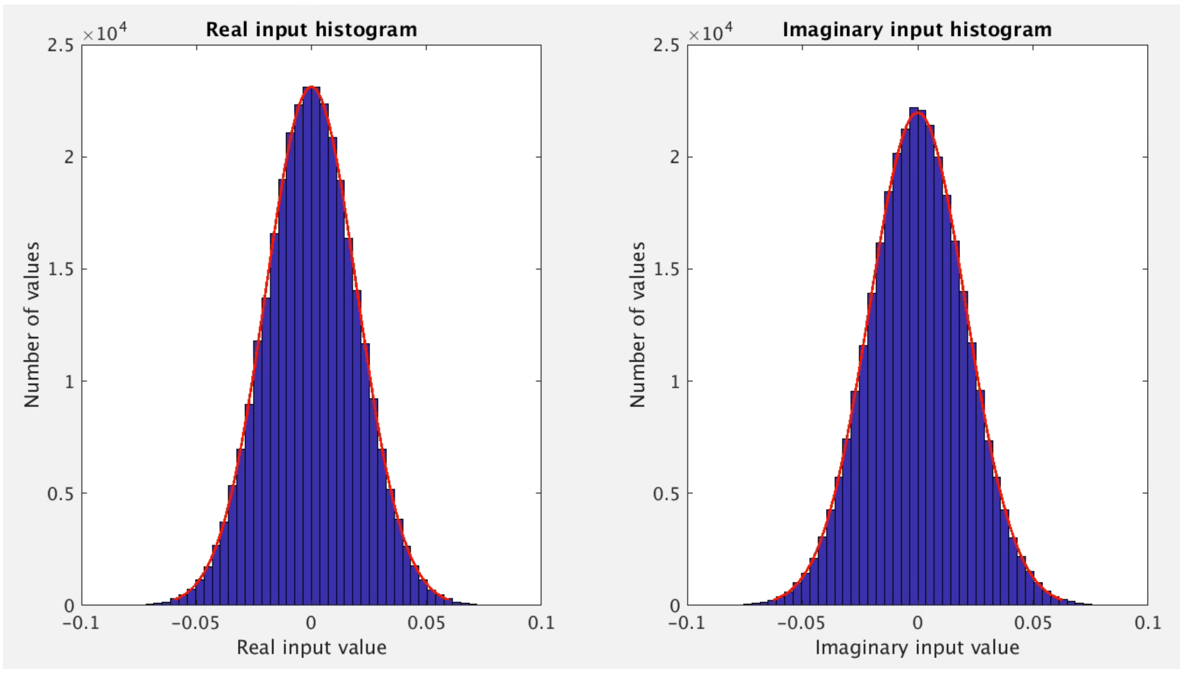


Figure 20. FFT raw input data, that was used in all FFT models when quantized to wanted format.

This same input was fed to all FFT models, and depending on the case, it was quantized to some format. Only in case of the reference FFT 2, the values stayed non-quantized throughout the FFT calculation. That reference added some error margin and it helped see all the properties of the chosen arithmetic representation.

The absolute value of complex input data was rarely close to zero or really far from zero, because both real and imaginary inputs were rarely very small or very big at the same time. Therefore, the absolute value of complex input data followed Rayleigh distribution, as can be seen in Figure 21, from the shape of the histogram.

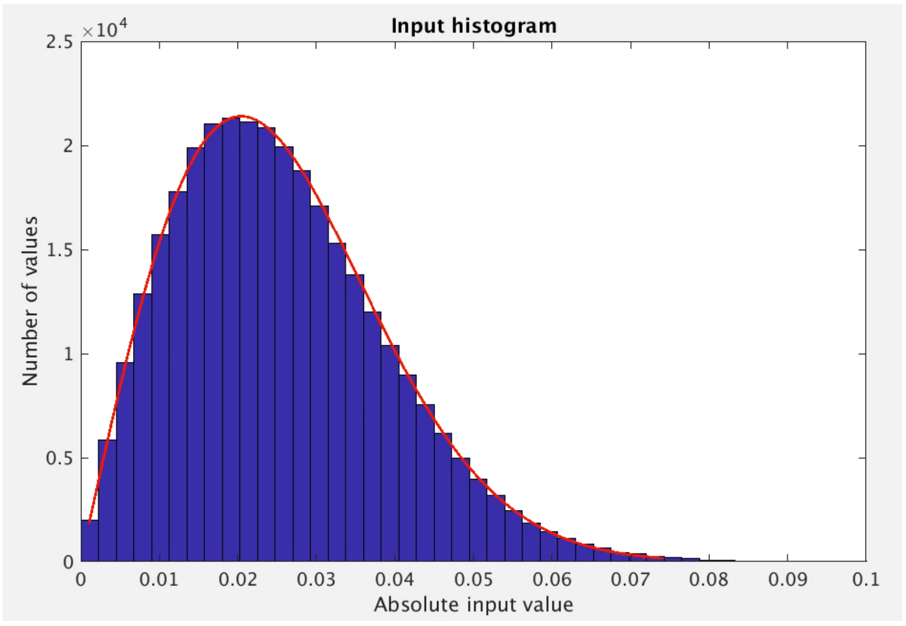


Figure 21. The absolute value of generated input, that followed the Rayleigh distribution.

4. SYNTHESIS

To get area and power reports for fixed point and floating-point adders and multipliers, synthesis was run for those arithmetic operations' Verilog and SystemVerilog models. The used synthesis tool was Design Compiler of Synopsys. CMUL did not need a separate model, because it consisted of real adders and multipliers. That allowed area calculation for CMUL without its own separate model.

The synthesis used a library that supports the chosen 12 nm technology. The synthesis was run for addition and multiplication operation for various word lengths, with clock frequency of 833 MHz. The same WLs were used in synthesis as in Matlab scripts, so that the area estimates could be calculated for similar FFTs that were implemented with Matlab.

Once addition and multiplication synthesis was executed, area results for FFT were calculated from separate operations. FFT consisted of one complex multiplication and two addition operations, and therefore the total arithmetic area of FFT was estimated using areas of four real multipliers and six adders. The used butterfly structure with synthesized real arithmetic operations is represented in Figure 22.

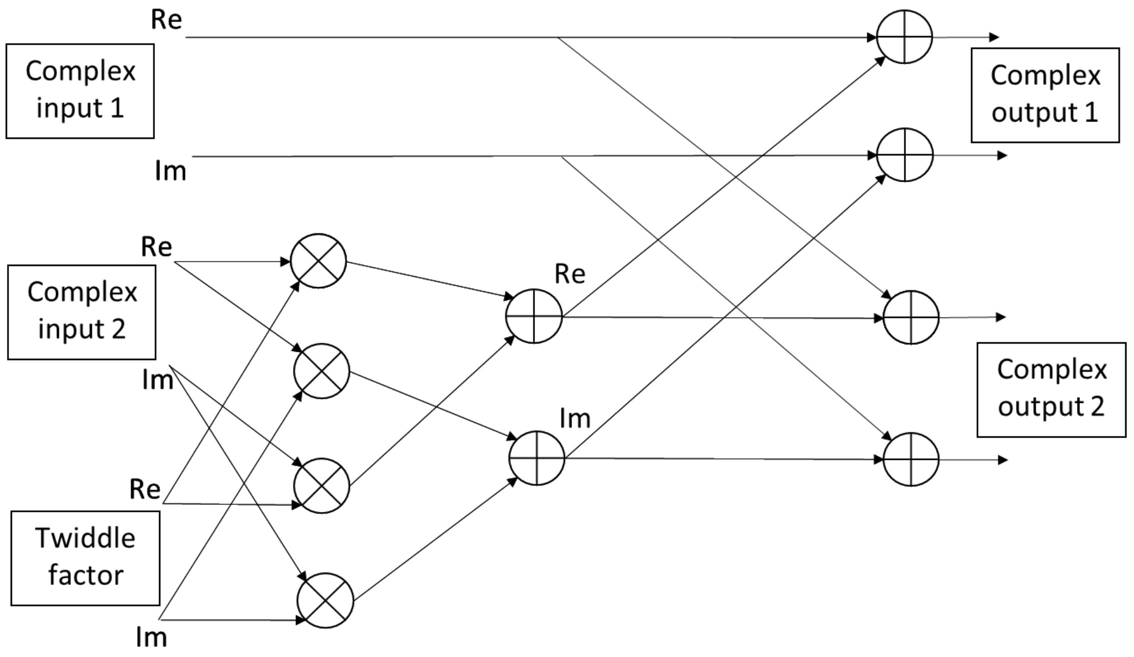


Figure 22. The FFT butterfly operation that consisted of six real adders and four multipliers.

The SystemVerilog models followed the addition and multiplication flowcharts represented in Chapter 2 with few exceptions. In these models, the word length did not change from the input word length at any stage because of the chosen iterative architecture. So arithmetic operation RTL-codes had the same number of bits in input and output. That of course led to some loss of precision in the result and a chance of overflow if the input values were not small enough. In addition, mantissas were always in two's complement format.

The block diagram of FXP ADD is represented in Figure 23 a), and its structure was very simple, because two's complement addition operation is straightforward. Figure 23 b) represents the block diagram of FXP MUL operation and, the structure seems simple, but two's complement multiplication is not as simple as sign + magnitude multiplication. The area-optimized flexible architecture of the signed MUL operation was chosen by the synthesis tool and is therefore not further explained here.

Figure 24 a) represents the block diagram of FLP ADD operation that allowed simple mantissa addition, because of used two’s complement format. Before addition, the mantissas were shifted according to difference in exponents. After mantissa addition, the result mantissa was normalized and exponent adjusted accordingly. Figure 24 b) represents FLP MUL operation, in which exponents were added and mantissas multiplied. The synthesis tool chose the component with the best area-delay relation, as in case of FXP, and that did not therefore distort the results of FXP-FLP comparison.

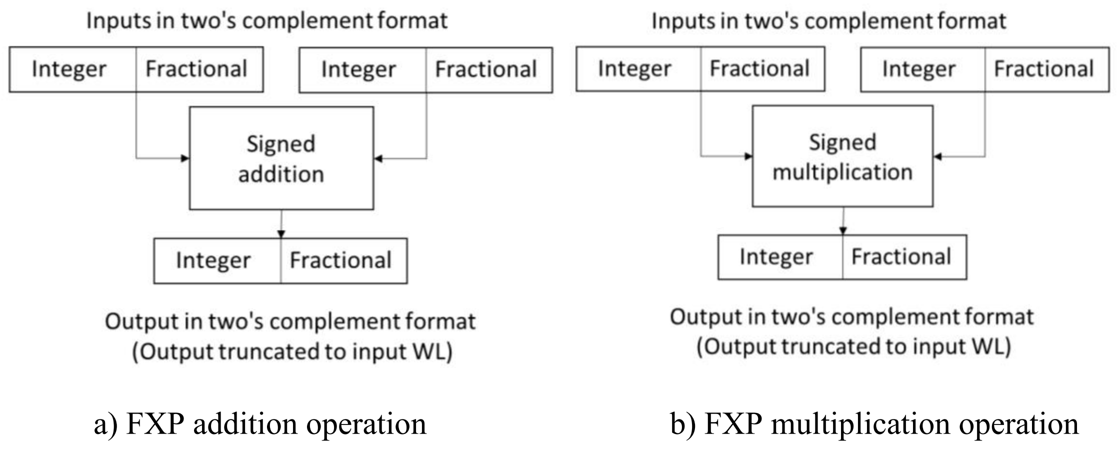

Inputs in two's complement format
Integer
Fractional
Integer
Fractional
Signed addition
Integer
Fractional
Output in two's complement format
(Output truncated to input WL)
Inputs in two's complement format
Integer
Fractional
Integer
Fractional
Signed multiplication
Integer
Fractional
Output in two's complement format
(Output truncated to input WL)
a) FXP addition operation
b) FXP multiplication operation

Figure 23. Block diagram of synthesized FXP arithmetic operations.

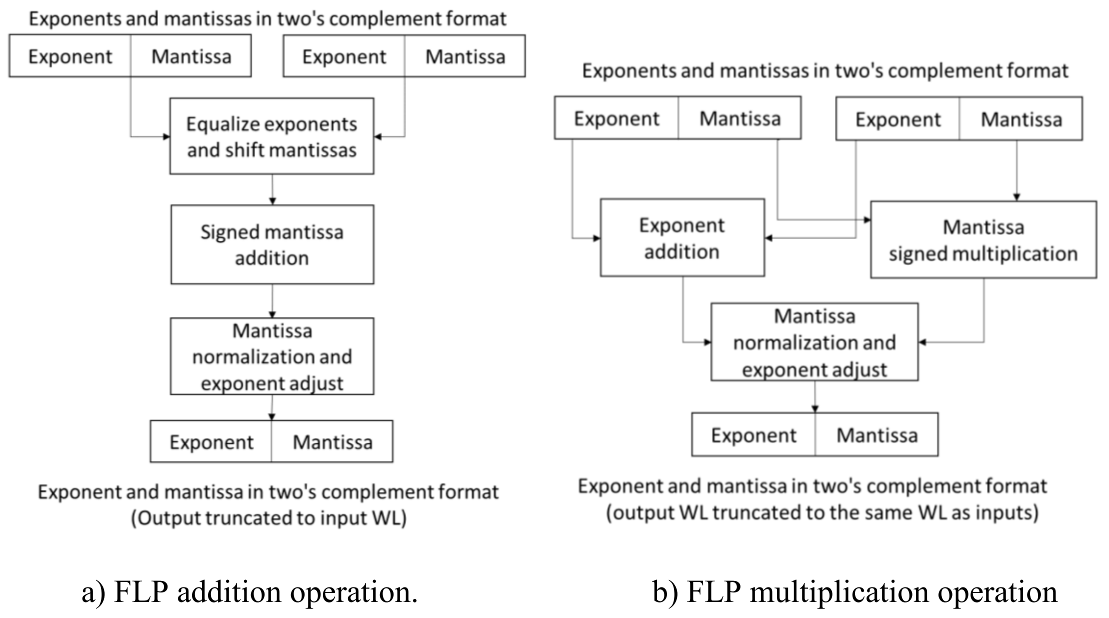

Exponents and mantissas in two's complement format
Exponent
Mantissa
Exponent
Mantissa
Equalize exponents and shift mantissas
Signed mantissa addition
Mantissa normalization and exponent adjust
Exponent
Mantissa
Exponent and mantissa in two's complement format
(Output truncated to input WL)
Exponents and mantissas in two's complement format
Exponent
Mantissa
Exponent
Mantissa
Exponent addition
Mantissa signed multiplication
Mantissa normalization and exponent adjust
Exponent
Mantissa
Exponent and mantissa in two's complement format
(output WL truncated to the same WL as inputs)
a) FLP addition operation.
b) FLP multiplication operation

Figure 24. Block diagram of synthesized FLP arithmetic operations.

5. RESULTS AND ANALYSIS OF FXP AND FLP COMPARISON

This chapter introduces the results of Matlab calculations and synthesis of arithmetic operations. Quantization models and their simulations were examined first to get clearer idea of the quantization process that later took place inside the FFT algorithm. This chapter provides all necessary information concerning conclusions done in next chapter. All calculations used all interesting word lengths. In FXPs case, the used word lengths were from nine to twenty-four, including sign bit and in case of FLP exponent word length varied from two to four and mantissa word length from six to sixteen. Mantissa part included its sign and therefore the total word length varied between eight and twenty.

5.1. Quantization

Using different WLs in quantization allowed error examination and dynamic range calculation for different formats, needed later in FFT algorithm part of the thesis. Quantization to interesting word lengths allowed error value and behavior studying.

First, absolute and relative error behavior was examined by sweeping the full range with small step. Quantization used only real values, because real and imaginary parts of complex input get the same quantization treatment separately.

In addition, the differences between floating-point implementations with and without gradual underflow were examined, to truly see the advantage of implementing gradual underflow. The FLP quantization model with gradual underflow was later implemented to FFT.

5.1.1. Dynamic range

The dynamic range (DR) of arithmetic representation is one feature to consider during comparison. In general, it requires less bits to get large dynamic range with floating point than with fixed-point, especially when FLP has gradual underflow implemented.

Figure 25 shows the plots of dynamic range of FXP. As can be seen in the figure, adding one bit increases the dynamic range by approximately 6 dB, as expected. Figure 26 illustrates the FLP's dynamic ranges with different word lengths, and as can be seen, only the number of exponent bits affected the dynamic range. Adding one exponent bit doubles the DR. In this case, gradual underflow was not implemented.

Figure 27 illustrates the effect of gradual underflow to dynamic range. Mantissa bit changes are on x-axis, but also the exponent changes. Mantissa bit changes led to 6 dB increase in DR and exponent change increased the DR more greatly. The DR values in Figure 26 set the minimum dynamic range of representation without the gradual underflow, to which gradual underflow added the effect of mantissa bits in Figure 27. The red line in the figures represents a reference of 72 dB DR, just to make comparison easier.

These dynamic range values were used in comparing different representations and to better analyze the area results. After all, the dynamic range of representation can be one of the given specifications: some applications require wide dynamic range, because their calculations include very small and very big values.

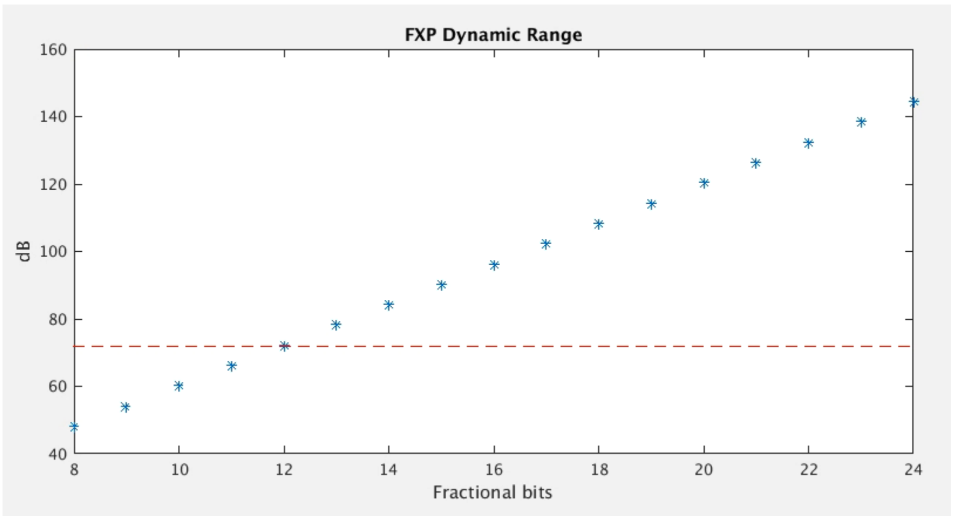


Figure 25. Dynamic ranges of FXP.

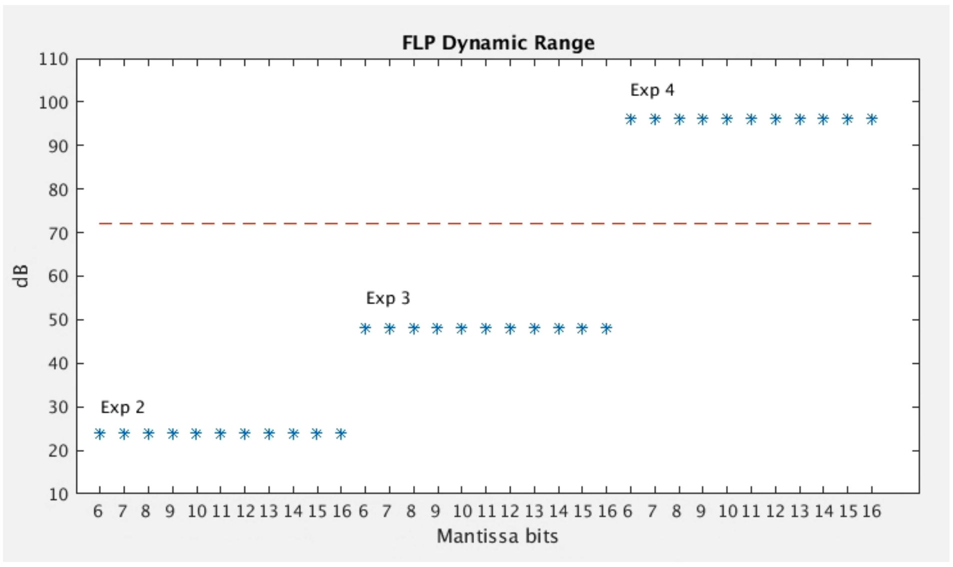


Figure 26. Dynamic ranges of FLP without gradual underflow.

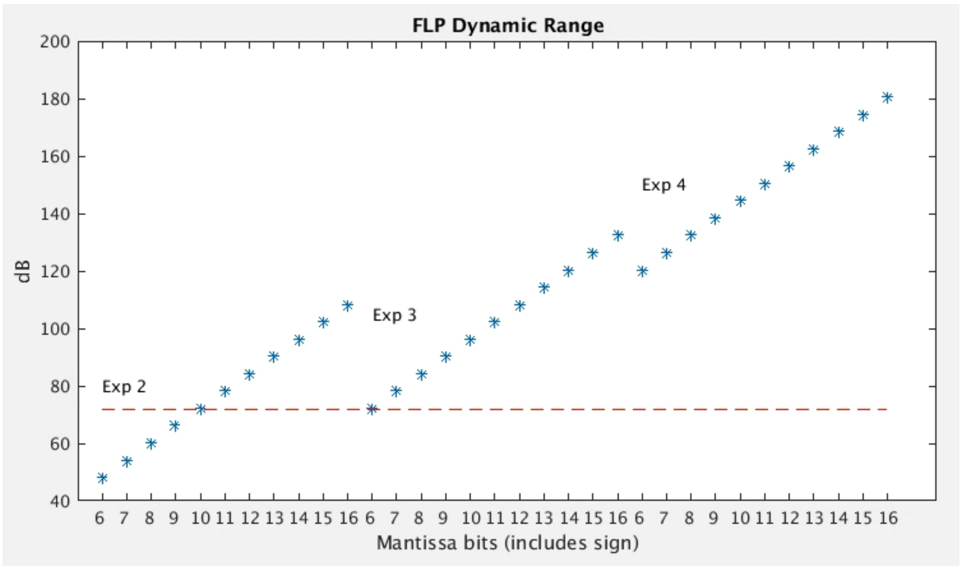


Figure 27. Dynamic ranges of FLP with gradual underflow.

Comparison of Figure 26 and Figure 27 gives great insight on the effect of gradual underflow, although it ignores the fact that the precision of smaller values gets worse when gradual underflow is applied. That was due to the mantissa working like FXP, and therefore it was possible that there are multiple zeros in front of meaningful bits, and that leads to less meaningful bits. Even though FXP was "too precise", when close to its maximum value, it had poor precision near zero.

5.2. FFT results

This chapter collects the results of Matlab calculations and synthesis of algorithm part of this thesis. Collecting the results helped in choosing the correct format for different specifications. SQNR results from Matlab were gone through first. Those SQNR results helped in area and power result analysis, because they gave a reference on the noise performance of the arithmetic representation in algorithm.

The input values followed the previously declared input data, and that was at its maximum nine percent of format's maximum to leave some room for the absolute value of real and imaginary part to increase. Figure 28 represents the output values of FFT, and the increase in magnitude was significant compared to input value. It can be seen, that there are some values that had absolute value of over 0.5, which is 0.1 in binary, and therefore the output fulfilled much bigger part of the range in each time it was run through. The input had to be restricted to such small part of range because of the iterative nature of chosen FFT algorithm, which did not allow the word length to increase. To avoid overflow, some headroom was left in input phase for the magnitude to grow when calculating.

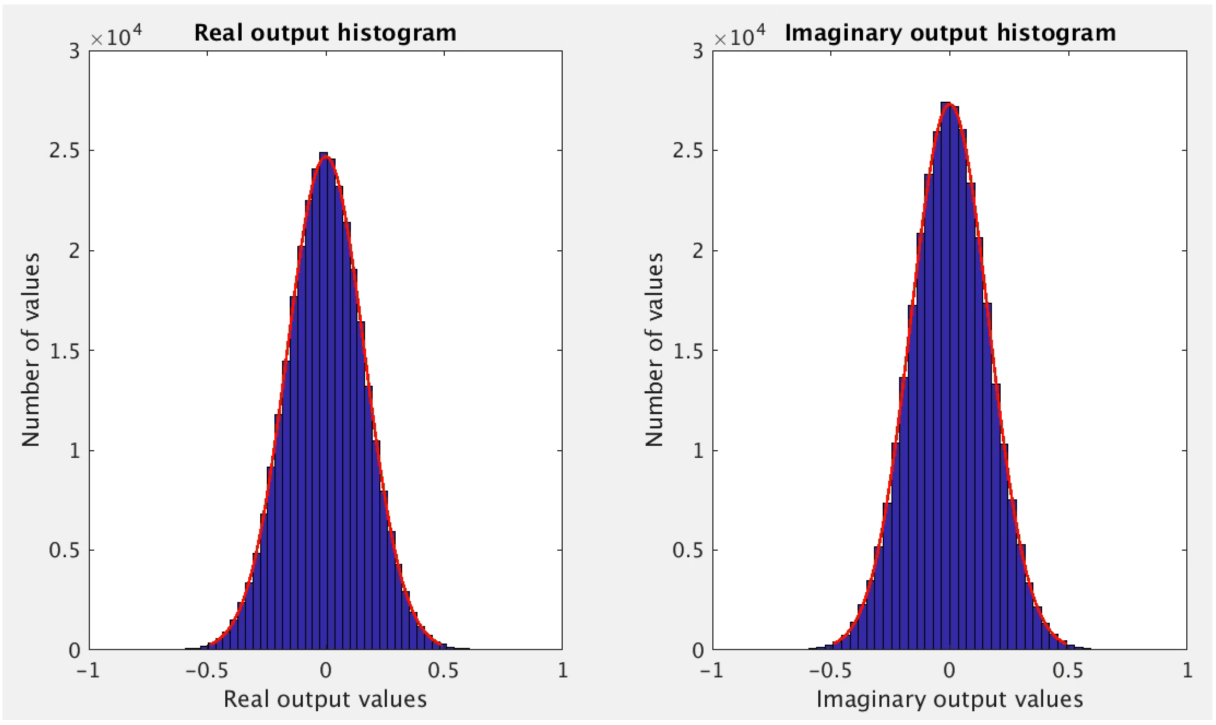


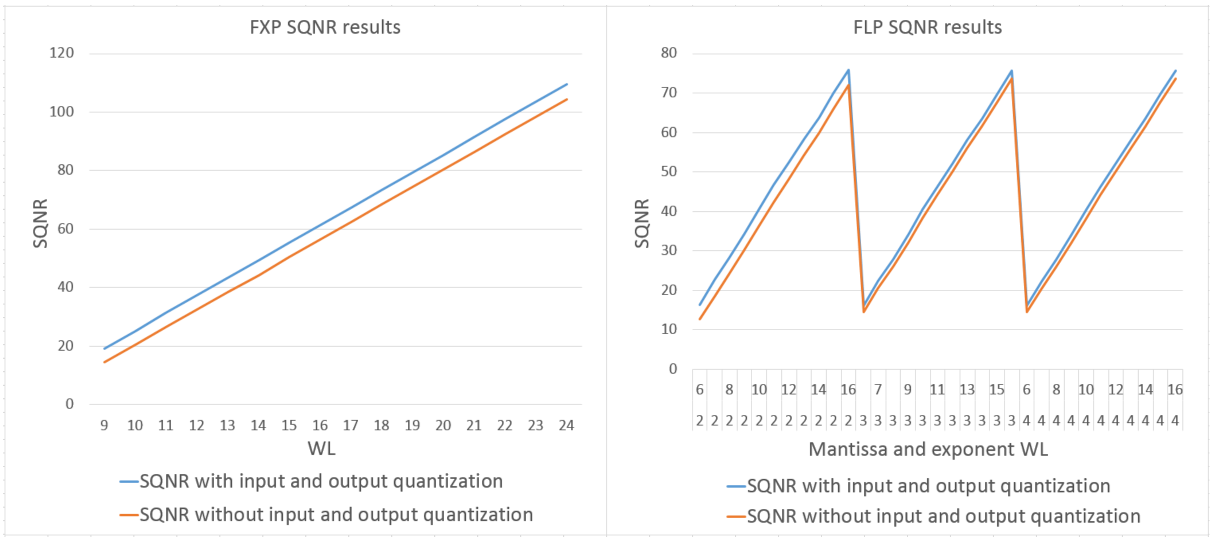
Figure 28. The real and imaginary output, that follow Gaussian distribution, and the values use bigger part of format's range.

5.2.1. SQNR of FFT

One main comparison method for FXP and FLP in this thesis was signal to quantization noise ratio (SQNR), because of its comparability and importance. The difference between reference FFT and the FFT model with chosen representation and word length allowed error calculation, which allowed the SQNR to be calculated.

All FXP FFT SQNR results are represented in Appendix 1, and those results are represented in graphical format in Figure 29 a). SQNR was calculated with and without reference quantization, and both results are represented in said figure. From those results, it can be seen that the SQNR increased approximately 6 dB when one bit was added, as did DR. Input was scaled so that there was no saturation during five thousand rounds that the FFT was run through. The SQNR results obtained without the reference quantization had worse SQNR values.

Appendix 2 collects the similar SQNR data about the FLP, and Figure 29 b) shows the results in graphical format. It should be taken into account that the x-axis is not linear, but shows the mantissa and exponent word lengths, going through all mantissa word lengths linearly with one exponent WL, before moving to next exponent WL.



a) FXP SQNR b) FLP SQNR

Figure 29. Graphical representation of SQNR results.

To rule out the formats that were not good enough, it was more reliable to use the SQNR results obtained without reference quantization in decision making, because it added some error margin. For FXP the error margin was constantly approximately 5 dB and for FLP it changed between 4 dB and 2 dB, decreasing when number of exponent bits increased from two to three or four. The larger drop in SQNR was caused by the poor accuracy and flush to zero of smaller values, when FXP or FLP with small exponent was used.

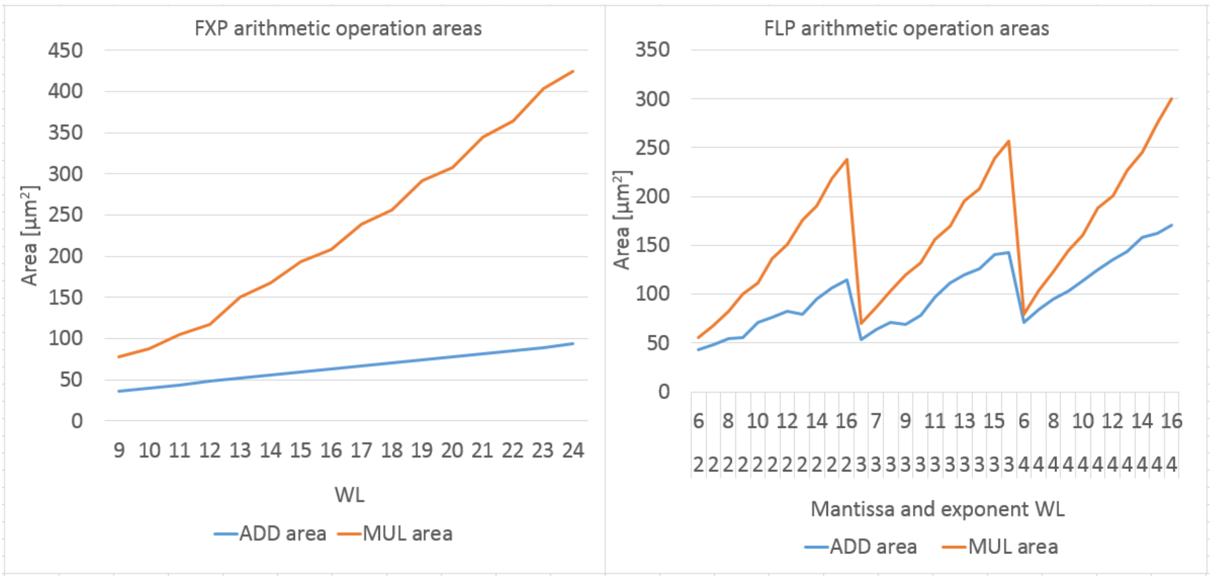
From SQNR results, it was concluded that FXP reached higher SQNR with same WL compared to FLP. To fulfill the set 40 dB SQNR requirement, FLP needed 13 bits and FXP needed 14 bits. In case of FLP, it was obvious that the exponent bits did not significantly increase the SQNR, and at least 11 mantissa bits were required for SQNR of 40 dB.

These SQNR results were used in area and power comparison, to better see the tradeoff between SQNR and area or power. The results showed that FXP had higher SQNR compared to FLP at all word lengths and their corresponding dynamic ranges. Because of the linearly increasing nature of SQNR, the behavior with other WLs can be estimated without running through all word lengths. For example, FXP with WL of 31 bits had DR of 180 dB, and its SQNR could be approximated to be 146 dB. Other mantissa WLs of FLP could be similarly approximated for chosen exponent value.

5.2.2. FFT area

The area of the chip was one of the key performance indicators of IC industry, and the culmination point of this thesis, as stated previously. After all, the quantization noise only determines the quantization noise performance of the format, and when finding the good-enough performance with FXP and FLP, the areas of said performances can be compared, and the smaller the area, the better.

Figure 30 a) represents the areas of FXP arithmetic operations and registers, and it was seen that the FXP multiplication operation was much bigger with similar word length compared to FXP addition operation. With WL of 9 bits, the multiplication operation was approximately twice as big as the addition, but with WL of 24 bits, the multiplication operation was over four times as big as addition operation. Some differences from linearity were probably caused by the compiler managing to optimize better with some word lengths. Those linearity exceptions happened more often in case of FLP, especially in addition operation, as can be seen in Figure 30 b), which represents the areas of arithmetic operations in case of FLP. FXP and FLP arithmetic areas also include registers.

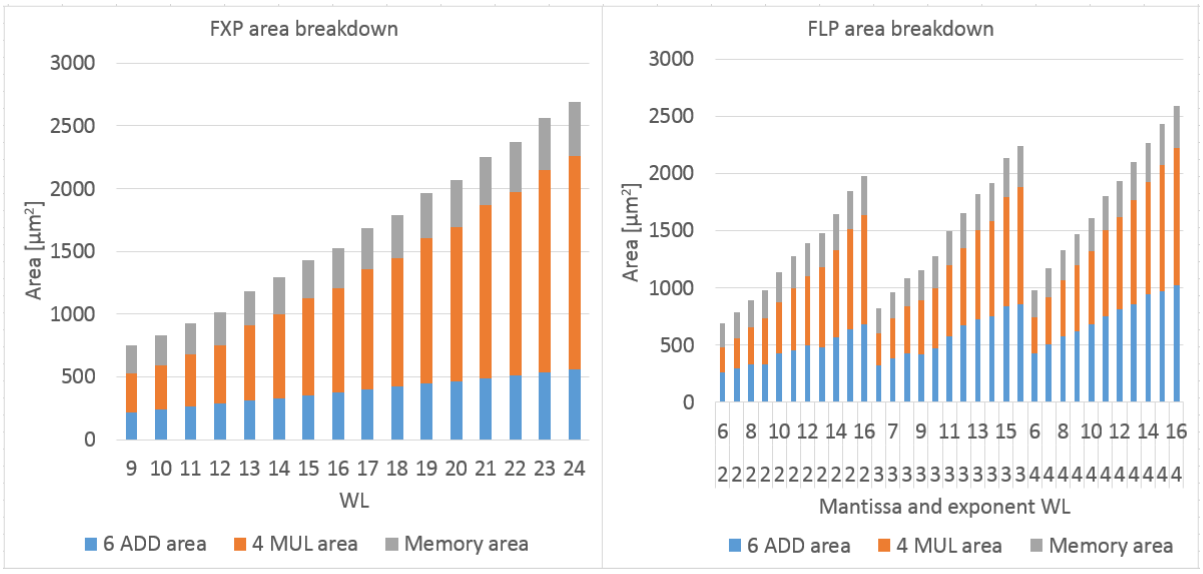


a) FXP operation areas b) FLP operation areas

Figure 30. The areas of addition and multiplication operations for FXP and FLP.

Therefore, it can be concluded that the area of ADD grew approximately linearly and the area of MUL grew more steeply when the WL increased. That was caused by the structures of the arithmetic operations. For example, in case of FXP, N-bit adder requires N 1-bit adders, and basic multiplier requires N^2 1-bit adders.

Arithmetic operations and registers made up the largest part of the total area of FFT. That arithmetic area consisted of six addition operations and four multiplication operations, as previously stated. Memory area was added to arithmetic area to make up the total area. The area division in FXP's case is represented in Figure 31 a), and it can be seen that the area grew approximately linearly when WL increased. The largest increase occurred in multiplication. Figure 31 b) represents the area breakdown of FLP FFT. Similar to previous, the x-axis of FLP graph was not linear, but represented the mantissa and exponent WLs. There were more nonlinearities in total areas in case of FLP, because of the nonlinearities in separate addition operation areas.



a) FXP areas breakdown

b) FLP area breakdown

Figure 31. The area breakdowns that show what the total areas consists of.

Figure 32 collects all area data of both arithmetic representations and SQNR results for analysis. The grey parts of areas represent the area values of the formats, whose SQNR did not fulfill the 40 dB specification. The SQNR results are represented with dots in corresponding color to area. Looking at just fulfilling the SQNR limit, FLP gave the smallest area, just slightly smaller than FXP, but FXP has slightly better SQNR. In dynamic ranges from 78-96 dB, there is no significant difference between FXP and FLP areas and SQNRs at similar DR. After dynamic range reached value 102 dB, FLP gave smaller areas than FXP. That led to some quantization-noise performance drop, but both fulfilled the specification. It was concluded that FLP gains advantage when big dynamic range is needed, as was expected.

Appendix 3 collects FXP area results and Appendix 4 collects the similar information about FLP, and those results were used to create these graphical representations of area data. The "FFT total" -categories include the memory area and arithmetic area that consisted of six adders and four multipliers. The arithmetic area also includes register area. To ensure scalability of the results, the memory area values were obtained by taking the estimate from larger, 512-word memory, and scaling it down to 64-word, by dividing the area by eight.

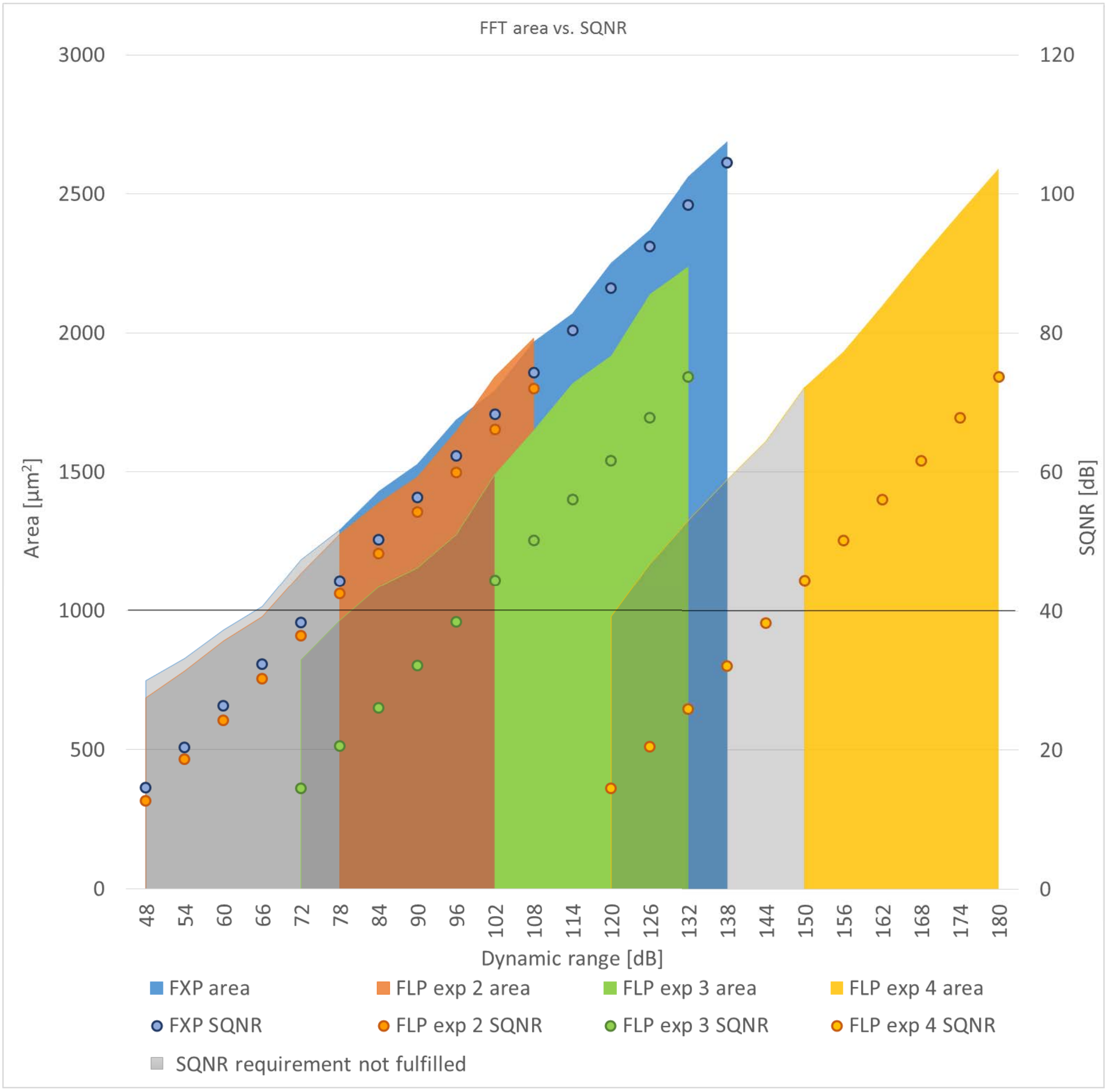


Figure 32. FXP and FLP total areas are collected here to make comparison and decision making easier.

5.2.3. *FFT power*

Power was not included in KPIs in this thesis, but was briefly gone through to see the correlation between SQNR, area and power. Assumption was that small area correlates with small power. The power results can offer some insight considering future work regarding the subject.

Synthesis reports from Synopsys Design Compiler provided the power results. Those results included the dynamic power consumed by arithmetic and registering, but not the power consumed by memory leakage and access. Appendix 5 and Appendix 6 collect the power results of fixed-point and floating-point FFTs.

Figure 33 represents the correlation between area and power in ADD and MUL operations. The x-axis shows the total WLs and first there are FLP results, first with two exponent bits, and increasing all the way to four exponent bits. All mantissa WLs are gone through for each exponent before moving to next exponent length. The last results are FXP results. From this representation of results, it was obvious that FXP had bigger power consumption compared to area. For example, in case of ADD, which is represented with blue. FXP required more power even though its area was significantly smaller than of FLPs, especially with longer WLs.

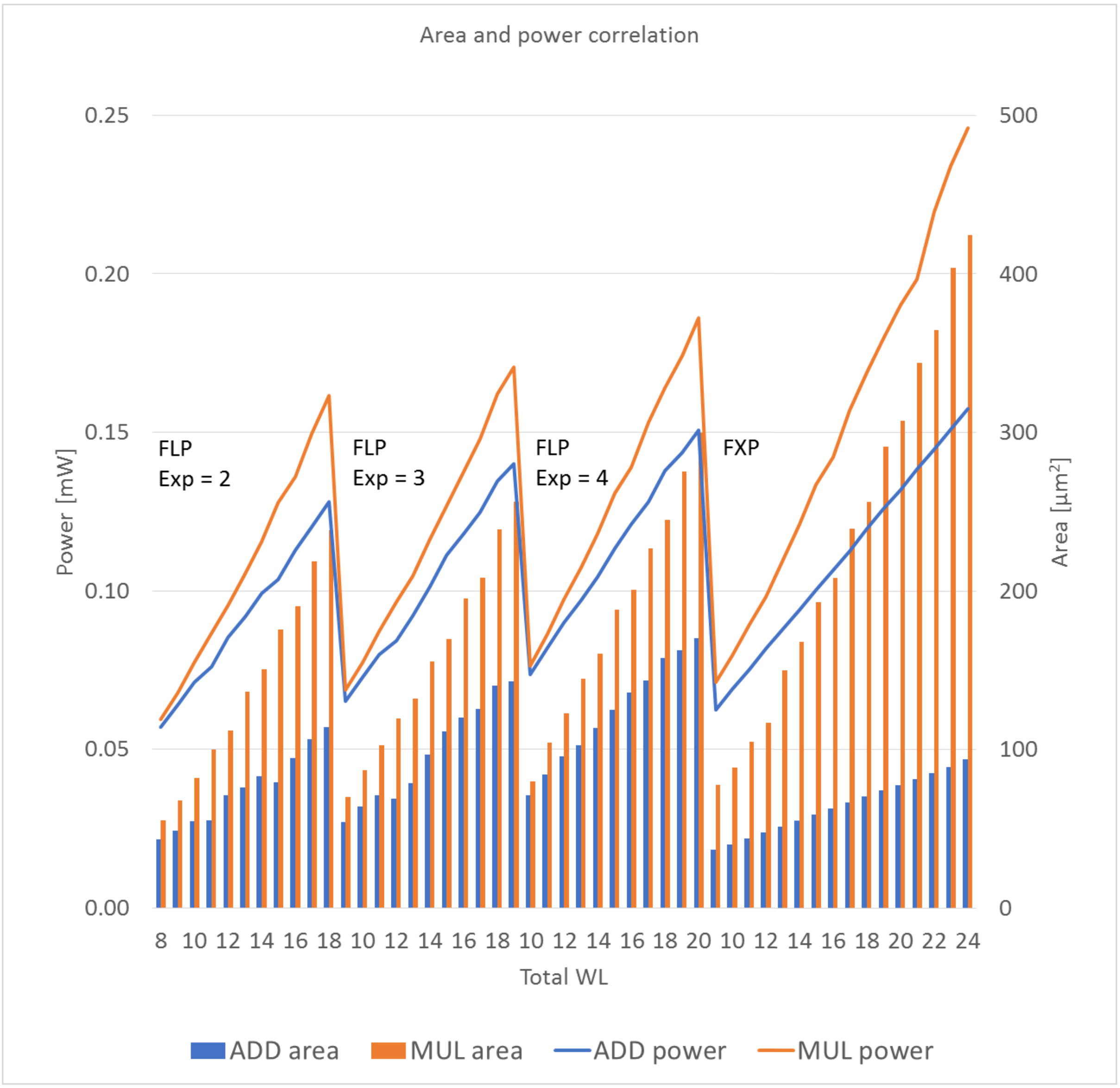


Figure 33. Area and power comparison of all used WLs of both representations, which shows the area and power correlation.

Figure 34 shows both power results in graphical format, for easier inspection. The x-axis is dynamic range, left y-axis is power and right y-axis is SQNR. As in previously introduced area figure, all power graph areas whose SQNR did not fulfill the 40 dB specification are represented in grey.

It can be seen that the shape of the figure correlates greatly with areas in Figure 32. The power of FXP was a bit higher, related to area, than in FLP's case. It was obvious, that the MUL power increased more in case of FXP, and that showed the correlation between area increase and power increase. On the other hand, the area correlation was not quite as clear in case of FLP. The powers of ADD and MUL operations were relatively similar with each other, as in area the difference was more significant.

In lower dynamic ranges, the power difference between FXP and FLP was not as significant as in higher DR values. In addition, especially in dynamic ranges below 102 dB, the SQNR was a little better in FXP than in FLP, so there was no significant difference between representations. After that, the power difference was more significant, and therefore FLP was the choice for dynamic ranges over 102 dB. Anyway, the power of FXP was higher, so in low-power applications, this should be considered in further detail.

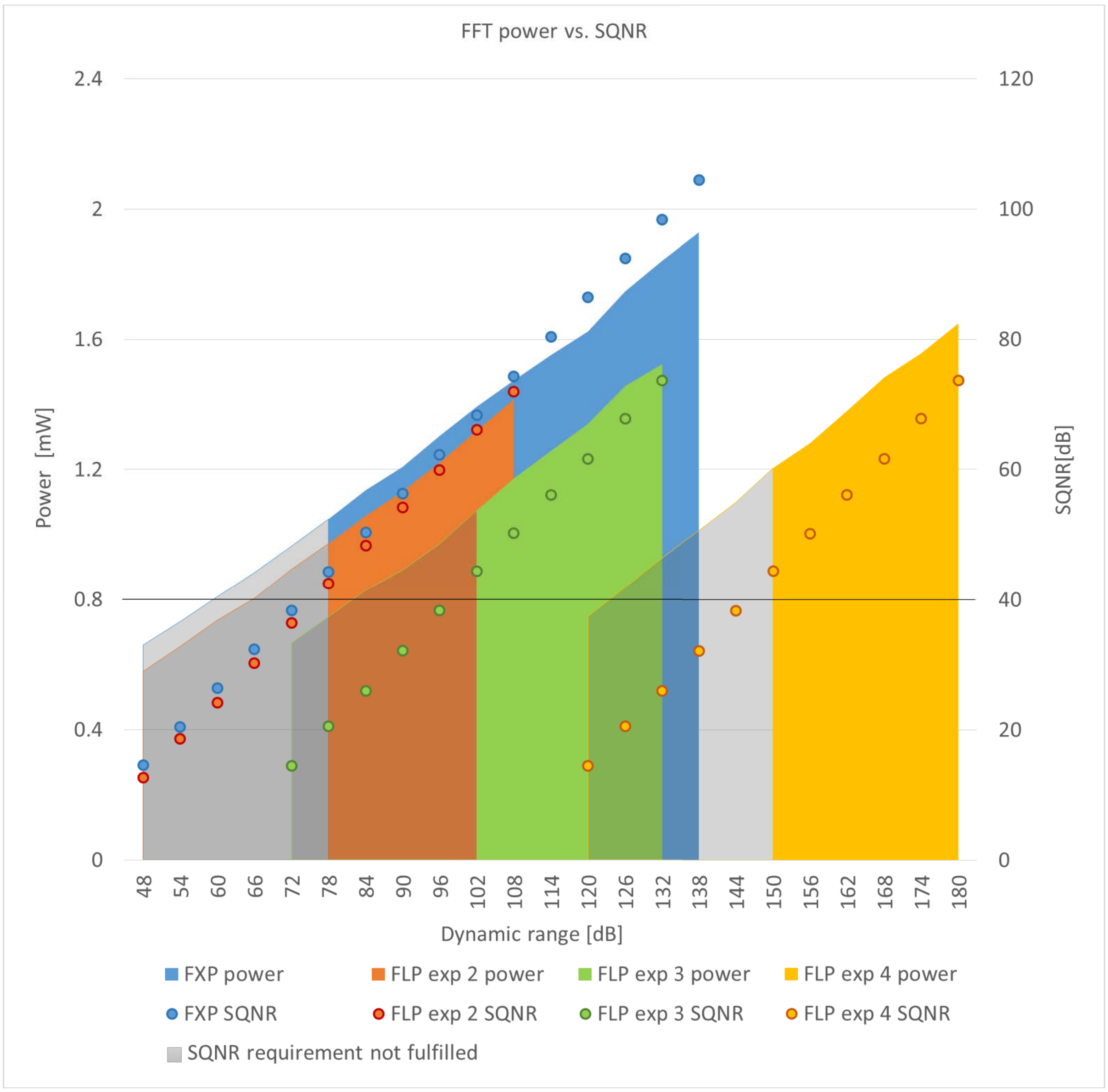


Figure 34. FFT power results for FXP and FLP.

6. CONCLUSION OF FXP AND FLP COMPARISON

This thesis compared two different numerical representations: fixed-point and floating-point by means of signal-to-quantization-noise-ratio (SQNR). After SQNR examination, area and power were taken under consideration. After all those are the features that affect the cost and usability of the product the most.

When concluding the results, it needed to be taken into account, that especially in case of FXP, the input limitation to 9 % of maximum value led to at least 3 zeros in the MSB side of the value. The word length had to stay the same throughout the whole algorithm due to iterative nature of the chosen memory-based FFT architecture.

The features taken into account when making the decision between FXP and FLP were SQNR, area and dynamic range. SQNR ruled out the unfit areas from the comparison and dynamic range added perspective. Power was also gone through briefly, but it did not affect the decision-making.

It was concluded that when dynamic range was less than 102 dB, there was no significant difference between FXP and FLP, even though a small tradeoff between slightly better SQNR and slightly larger area needed to be considered. With dynamic range of over 102 dB, the advantage of FLP became clearer.

The benefit of FLP got clearer when exponent had at least 3 bits. FLP with exponent of two bits had just a little advantage on area and power when compared to FXP in small dynamic ranges and a small disadvantage from SQNR point of view. When using more exponent bits and therefore bigger DR, the advantage of FLP became clearer. Of course, the SQNR drop at same dynamic range was big when compared to FXP, but the area advantage was huge. For example, the area was approximately twice as big in FXP compared to FLP with exponent of four bits, at similar dynamic ranges. There were no FXP results for DRs of over 138 dB, but the trend was seen from obtained results: the FXP area and SQNR grew approximately linearly.

If only looking at fulfilling the SQNR specification without any dynamic range preference, FLP gave the smallest area, but the difference between FXP and FLP was negligible. Nevertheless, that did not leave much room for unexpected events, because the DR was small, and SQNR was just a little bit above 40 dB. Wider DR works as a safety feature that helps the design handle unexpected events. Bigger DR has some headroom that prevents SQNR drop caused by saturation.

In addition, it should be kept in mind, that SQNR of 100 dB is basically never needed, because some other feature would anyway cause some noise to the system and drop the SNR below 100 dB. The same area-SQNR figure is also valid with different SQNR specification from 40 dB. With different SQNR requirements, the grey areas in Figure 32 and Figure 34 would not be reliable, and those should be changed to see the areas that fulfill the set requirement.

7. DISCUSSION

In this chapter, the reliability of the results is discussed and some speculations about different outcomes are made. For example, the effect of input data caused some disturbances in the comparison process. Floating-point representation's big dynamic range and especially its advantages in both ends of the range could give it a great advantage compared to fixed-point at equal inputs. The goal was to make the comparison as fair as possible, so the input data needed to be chosen in a way, that it did not give a significant advantage for either of the arithmetic representations. It was the goal of this thesis to see which representation is better in regular case, and to find out if there was some sweet spot for either fixed-point or floating-point arithmetic.

For the chosen FFT-algorithm the goal was achieved, and it was found that at dynamic ranges less than 102 dB, there was no significant difference in area and quantization noise performance between FXP and FLP. At wider dynamic ranges, the FLP started to gain area advantage, but a tradeoff was made between great signal to quantization noise ratio and smaller area. The area advantage was great enough, and it was most important that the SQNR fulfilled the requirement, not to get as high as possible SQNR. Therefore, with larger than 102 dB dynamic range, floating-point representation showed great advantage in both area and power. Overall, the performance was better in floating-point representation.

The results of this thesis were somewhat similar to results in the previous work from Tampere University, when focusing on signal to noise ratio. [13] In that research it was concluded, that with larger amount of exponent bits in FLP, that would be the choice, but when large exponent field is not possible, the choice would be FXP. The difference between this thesis and that previous research was that in this thesis, a significant advantage of using FXP was not found. This work agreed mostly on the previous works introduced previously, but FLP had the power advantage at all dynamic ranges. There were some conflicts between previous works regarding power, but FXP was mostly thought to be more power efficient than FLP, which was not the case in this thesis.

One way to improve the study would be to find the optimal input scaling for each representation. While looking for the value that did not cause overflow, it was noticed that FXP benefited of as large as possible input values, and FLP of smaller input values. Therefore, input scaling optimization could be used to find the best possible signal to quantization noise ratios for FXP and FLP with different number of exponent bits. In addition, it would probably be beneficial to test the same FFT models with a real signal, for example sinusoidal. That might affect the results, because then the output would not be noise, but instead some spikes caused by the sinusoidal frequencies. Sinusoidal input would allow better examination of quantization noise spectrums, which could be interesting because in case of floating-point, the quantization noise depends heavily on the signal magnitude.

The used FFT algorithm could also be better optimized so that the input values could use bigger part of the range. That could be done for example with some bit shifting in necessary stages of calculation. For example, in case of radix-2 butterfly, the outputs could be divided by two, which equals one right shift. That would prevent the output from growing out of range during calculations. The precision loss caused by shifting should be studied.

As future work it could be examined how a more addition heavy algorithm affects the results, and on the contrary a multiplication heavy algorithm. Because the areas of operations depend heavily on used representation, for example, FLP multiplication is significantly smaller than FXP multiplication; it would be interesting to study the advantages of performing some arithmetic operations with different arithmetic representation. That would require studying the conversion block and its area, power consumption and the performance drop caused by it. Then those results could be compared to just performing the operation in the other format. Therefore, it would be beneficial to further study if big multiplication operations in FXP design should be performed in FLP for smaller area. A sweet spot for FLP multiplication in FXP design could possibly be found in further studying of the subject.

Overall, floating-point representation offers great dynamic range, which allows carefree algorithm development especially in case of longer algorithm pipeline. Cheaply achieved big dynamic range gives the design a greater possibility to succeed with fast development. It could of course be possible that fixed-point representation would give smaller area for the algorithm when word lengths are correctly optimized at every point of the algorithm, but that takes a lot of time from the developer. If floating-point representation is used, it is possible that the full algorithm can be developed using only one word length throughout the design. With sufficient dynamic range, the design can tolerate and avoid unexpected events to keep the signal-to-quantization-noise-ratio acceptable.

8. SUMMARY

This thesis compared fixed-point and floating-point arithmetic representations in terms of area required by chosen memory-based 64-point radix-2 FFT-algorithm. SQNR of 40 dB was set as a requirement that must be fulfilled by the representation with current word length and dynamic range. The used WLs in case of FXP were 9-24 bits and in case of FLP 6-20 bits.

First, this thesis provided some necessary background theory about the arithmetic representations and the necessary arithmetic operations. A reference model was required for SQNR calculation over the algorithm. The used reference model was FFT-function of Matlab and those calculations were executed with double precision floating-point format.

FXP and FLP required their own quantization models that were first studied separately. Those models were then applied inside the FFT algorithm and the quantization noise was examined. Reference models were tried out with and without input and output quantization, but the results from the reference without the quantization were used in the decision-making, because of the added error margin compared to the one with the reference quantization.

The chosen FFT-algorithm was of iterative nature, and therefore the word lengths inside it were fixed. Because the word length did not increase inside the algorithm, the input signal needed to be restricted so that overflow did not occur during calculations. This gave some advantage for FLP, because of its larger dynamic range. The input maximum amplitude was normalized to 9 % of the format's full range after an iterative search for input that did not cause an overflow during five thousand FFT runs in Matlab.

Addition and multiplication operations had their own SystemVerilog RTL-models that were synthesized to obtain area results. Because of the chosen FFT architecture, only one radix-2 butterfly structure was needed, and that consisted of six addition operations and four multiplications. All separate operations used real values, even though the calculations in FFT use complex values. Arithmetic and register area was estimated by summing up six adder areas and four multiplier areas of same word length. The area required by memory was then added to area estimate. It was obvious from the area results that the addition operation required larger area when implemented with FLP and multiplication operation required larger area when FXP was used. Power estimate was obtained in similar way, but the memory power consumption was not included in the estimation. FLP held the power advantage at all dynamic ranges.

From collected SQNR and area results for different word lengths and dynamic ranges, it was possible to compare the areas and see which word lengths fulfilled the SQNR requirements, and were included in comparison. It was seen from the results that FXP and FLP with two exponent bits had just slight area and SQNR difference.

In conclusion, when small dynamic range was enough, there was no significant advantage for either of the representations, when the SQNR requirement was fulfilled. FXP had slightly better SQNR and slightly larger area compared to FLP at less than 102 dB dynamic range. When FLP had at least three exponent bits, providing dynamic ranges of over 102 dB, FLP had a clear area advantage over FXP.

9. REFERENCES

[1] Baher, H. (1990) Analog & Digital Signal Processing, Wiley & Sons, New Jersey, 247-264 p.

[2] Cavanagh, J.J.F. (1984) Digital Computer Arithmetic: Design and Implementation, McGraw-Hill College, New York, 98-233 p., 353-390p., 426-428 p.

[3] Waser, S., Flynn, M. (1982) Introduction to Arithmetic for Digital System Designers, CBS College Publishing, New York, 12-27 p., 105-106 p., 206-211 p.

[4] Stallings, W. (2003) Computer Organization & Architecture, Eighth edition, Pearson Education, London, 308-341 p.

[5] Kuo, S.M., Lee, B.H. (2001) Real-Time Digital Signal Processing: Implementations and Applications, John Wiley & Sons, New Jersey, 98-104 p., 157-160 p., 303-331 p.

[6] Yates, R. (read 8.4.2019) Fixed-point Arithmetic: An Introduction, URL: https://courses.cs.washington.edu/courses/cse467/08au/labs/l5/fp.pdf

[7] Twin, A. (read 22.3.2019) Key Performance Indicators (KPI) URL: https://www.investopedia.com/terms/k/kpi.asp

[8] Chip Basics: Time, area, Power, Reliability and Configurability. (read 22.3.2019) URL: https://www.doc.ic.ac.uk/~wl/teachlocal/cuscomp/notes/chapter2.pdf

[9] H., Liu, G., Li (2005) OFDM-Based Broadband Wireless Networks: Design and Optimization, John Wiley & Sons, New Jersey, 17-29 p.

[10] Goldberg D. (1991) What Every Computer Scientist Should Know About Floating-Point Arithmetic, Computing Surveys, Issue 1991.

[11] IEEE Standard for Floating-Point Arithmetic (IEEE 754) (read 12.3.2019) URL: https://ieeexplore.ieee.org/document/4610935

[12] Werther, O., Minihold, R. (read 28.3.2019) LTE: System Specifications and Their Impact on RF & Base Band Circuits Application note. URL: https://cdn.rohde-schwarz.com/pws/dl_downloads/dl_application/application_notes/1ma221/1MA221_0e.pdf

[13] Ghalib, A. (2011) Analysis of Fixed-Point and Floating-Point Quantization in Fast Fourier Transform, Tampere University of Technology, Tampere.

[14] Chang, W.-H., Nguyen, T. (2008) On the Fixed-Point Accuracy Analysis of FFT Algorithms, University of California San Diego, La Jolla

[15] Barrois, B., Sentieys, O. (2017) Customizing Fixed-Point and Floating-Point Arithmetic- A case study in K-means clustering, SiPS 2017 - IEEE International Workshop on Signal Processing Systems, Lorient

[16] Barrois, B. (2017) Methods to Evaluate Accuracy-Energy Trade-Off in Operator-Level Approximate Computing, University of Rennes, Rennes

[17] Duckett, G., Pennington, T. (read 4.3.2019) Fixed-Point vs. Floating-Point DSP for Superior audio. URL: https://web.archive.org/web/20060515074349/http:/www.rane.com/note153.html

[18] Bhattacharyya, S.S., Deprettere, E.F., Leupers, R.,Takala, J. (2010) Handbook of Signal Processing systems, Springer Publishing Company, New York

10. APPENDICES

Appendix 1. FXP SQNR results.

Format	WL	Dynamic range [dB]	SQNR [dB] (with reference quantization)	SQNR [dB] (without reference quantization)
S0.8	9	48	18.98	14.52
S0.9	10	54	25.18	20.39
S0.10	11	60	31.30	26.35
S0.11	12	66	37.34	32.31
S0.12	13	72	43.38	38.32
S0.13	14	78	49.31	44.27
S0.14	15	84	55.36	50.31
S0.15	16	90	61.36	56.30
S0.16	17	96	67.35	62.30
S0.17	18	102	73.36	68.31
S0.18	19	108	79.39	74.34
S0.19	20	114	85.38	80.35
S0.20	21	120	91.49	86.42
S0.21	22	126	97.51	92.43
S0.22	23	132	103.46	98.41
S0.23	24	138	109.56	104.48

Appendix 2. FLP SQNR results

WL			Dynamic range [dB]	SQNR [dB] (with reference quantization)	SQNR [dB] (without reference quantization)
Exponent bits	Mantissa Bits	Total WL			
2	6	8	48	16.41	12.69
2	7	9	54	22.70	18.60
2	8	10	60	28.14	24.19
2	9	11	66	34.35	30.25
2	10	12	72	40.67	36.41
2	11	13	78	46.73	42.45
2	12	14	84	52.47	48.31
2	13	15	90	58.32	54.21
2	14	16	96	63.80	59.92
2	15	17	102	70.04	66.07
2	16	18	108	75.84	71.97
3	6	9	114	16.21	14.48
3	7	10	78	22.46	20.51
3	8	11	84	27.91	25.99
3	9	12	90	34.11	32.12
3	10	13	96	40.44	38.35
3	11	14	102	46.49	44.39
3	12	15	108	52.25	50.19
3	13	16	114	58.09	56.07
3	14	17	120	63.58	61.66
3	15	18	126	69.82	67.86
3	16	19	132	75.62	73.72
4	6	10	138	16.21	14.45
4	7	11	126	22.46	20.49
4	8	12	132	27.91	25.97
4	9	13	138	34.11	32.09
4	10	14	144	40.44	38.32
4	11	15	150	46.49	44.36
4	12	16	156	52.25	50.16
4	13	17	162	58.09	56.05
4	14	18	168	63.58	61.64
4	15	19	174	69.82	67.84
4	16	20	180	75.62	73.70

Appendix 3. Area results of FXP, that include the areas of arithmetic operations, memory, and whole FFT block.

Format	WL	Dynamic range [dB]	ADD Area [μm^2]	MUL Area [μm^2]	FFT arithmetic area (6 ADD + 4 MUL) [μm^2]	FFT memory area [μm^2]	FFT total area [μm^2]
S0.8	9	48	36.42	77.30	527.72	220.75	748.47
S0.9	10	54	40.18	88.18	593.80	234.50	828.30
S0.10	11	60	43.94	104.58	681.90	248.3375	930.28
S0.11	12	66	47.70	116.67	752.88	262.125	1015.01
S0.12	13	72	51.46	149.82	908.04	275.875	1183.92
S0.13	14	78	55.22	167.73	1002.24	289.625	1291.87
S0.14	15	84	58.98	193.90	1126.24	303.375	1429.62
S0.15	16	90	62.74	208.50	1210.44	317.25	1527.69
S0.16	17	96	66.50	239.36	1356.44	331.00	1687.44
S0.17	18	102	70.26	256.32	1446.84	344.75	1791.59
S0.18	19	108	74.02	291.26	1609.16	358.50	1967.66
S0.19	20	114	77.78	307.59	1697.04	372.25	2069.29
S0.20	21	120	81.54	344.13	1865.76	386.125	2251.89
S0.21	22	126	85.30	364.51	1969.84	399.875	2369.72
S0.22	23	132	89.06	403.55	2148.56	413.625	2562.19
S0.23	24	138	93.63	424.64	2260.18	427.375	2687.56

Appendix 4. Area results of FLP, that include the areas of arithmetic operations, memory, and whole FFT block.

WL		DR [dB]	ADD Area [µm^2]	MUL Area [µm^2]	FFT arith. area [µm^2] (4MUL + 6 ADD)	FFT memory area	FFT total area
Exponent bits	Mantissa Bits						
2	6	48	43.24	55.07	479.75	207.900	686.75
2	7	54	48.51	67.79	562.25	220.75	783.00
2	8	60	54.74	81.99	656.40	234.50	890.90
2	9	66	55.15	99.75	729.91	248.38	978.28
2	10	72	70.74	111.96	872.28	262.13	1134.40
2	11	78	75.79	136.51	1000.78	275.89	1276.66
2	12	84	82.80	150.33	1098.10	289.63	1387.73
2	13	90	79.29	175.69	1178.54	303.38	1481.92
2	14	96	94.59	190.22	1328.43	317.25	1645.68
2	15	102	106.35	218.42	1511.79	331.00	1842.79
2	16	108	114.06	238.07	1636.61	344.75	1981.36
3	6	72	53.90	69.78	602.51	220.75	823.26
3	7	78	64.00	86.63	730.50	234.50	965.00
3	8	84	71.07	102.67	837.11	248.38	1085.48
3	9	90	68.97	119.62	892.33	262.13	1154.45
3	10	96	78.56	131.75	998.35	275.88	1274.23
3	11	102	96.66	155.57	1202.21	289.63	1491.83
3	12	108	111.37	169.65	1346.79	303.38	1650.16
3	13	114	120.10	195.19	1501.40	317.25	1818.65
3	14	120	125.71	208.32	1587.51	331.00	1918.51
3	15	126	140.08	238.73	1795.42	344.75	2140.17
3	16	132	142.74	256.13	1880.95	358.50	2239.45
4	6	120	71.07	79.59	744.80	234.50	979.30
4	7	126	84.01	104.10	920.49	248.38	1168.87
4	8	132	95.33	122.98	1063.90	262.13	1326.02
4	9	138	102.85	144.76	1196.16	275.88	1472.04
4	10	144	113.21	160.54	1321.43	289.63	1611.05
4	11	150	125.01	188.19	1502.80	303.38	1806.17
4	12	156	135.62	200.61	1616.19	317.25	1933.44
4	13	162	143.33	227.01	1768.00	331.00	2099.00
4	14	168	157.70	244.70	1925.04	344.75	2269.79
4	15	174	162.24	275.23	2074.34	358.50	2432.84
4	16	180	170.16	299.67	2219.66	372.25	2591.91

Appendix 5. Fixed-point total powers of arithmetic operations.

Format	Total WL	ADD total power [mW]	MUL total power [mW]	FFT total arithmetic power [mW]
S0.8	9	0.06258	0.07119	0.66026
S0.9	10	0.06887	0.07975	0.73223
S0.10	11	0.07518	0.08942	0.80875
S0.11	12	0.08168	0.09825	0.88306
S0.12	13	0.08786	0.10940	0.96475
S0.13	14	0.09401	0.12090	1.04765
S0.14	15	0.10040	0.13350	1.13640
S0.15	16	0.10660	0.14220	1.20840
S0.16	17	0.11260	0.15700	1.30360
S0.17	18	0.11960	0.16890	1.39320
S0.18	19	0.12600	0.17950	1.47400
S0.19	20	0.13180	0.19020	1.55160
S0.20	21	0.13850	0.19830	1.62420
S0.21	22	0.14470	0.21970	1.74700
S0.22	23	0.15090	0.23390	1.84100
S0.23	24	0.15750	0.24590	1.92860

Appendix 6. Floating-point total powers of arithmetic operations.

WL			ADD total power [mW]	MUL total power [mW]	FFT total arithmetic power [mW]
Exponent	Mantissa	Total WL			
2	6	8	0.05695	0.05937	0.57917
2	7	9	0.06409	0.06782	0.65585
2	8	10	0.07112	0.07738	0.73621
2	9	11	0.07618	0.08670	0.80387
2	10	12	0.08541	0.09529	0.89364
2	11	13	0.09185	0.10540	0.97271
2	12	14	0.09914	0.11570	1.05765
2	13	15	0.10370	0.12780	1.13340
2	14	16	0.11290	0.13610	1.22180
2	15	17	0.12020	0.14970	1.32000
2	16	18	0.12820	0.16170	1.41600
3	6	9	0.06517	0.06884	0.66634
3	7	10	0.07292	0.07735	0.74692
3	8	11	0.07994	0.08726	0.82865
3	9	12	0.08427	0.09637	0.89110
3	10	13	0.09218	0.10480	0.97225
3	11	14	0.10150	0.11650	1.07500
3	12	15	0.11110	0.12660	1.17300
3	13	16	0.11790	0.13750	1.25740
3	14	17	0.12470	0.14810	1.34060
3	15	18	0.13460	0.16220	1.45640
3	16	19	0.14020	0.17070	1.52400
4	6	10	0.07376	0.07642	0.74820
4	7	11	0.08201	0.08642	0.83774
4	8	12	0.08995	0.09730	0.92893
4	9	13	0.09707	0.10750	1.01241
4	10	14	0.10450	0.11810	1.09940
4	11	15	0.11350	0.13080	1.20420
4	12	16	0.12090	0.13890	1.28100
4	13	17	0.12800	0.15310	1.38040
4	14	18	0.13780	0.16400	1.48280
4	15	19	0.14350	0.17400	1.55700
4	16	20	0.15070	0.18610	1.64860