Preserving dynamic range in fixed-point representation in 5G

Master of Science Thesis
University of Turku
Department of Future Technologies
NOKIA
2020
Husam Hreitani

Acknowledgement

I would like to thank my Family, Line-Manager Jaako Maunuksela, Supervisor Mohammadhashem Haghbayan and Instructors: Mohammad Fattaholmannan, Hamidreza Shariatmadari, Shahriar Shahabuddin, Project Manager Dario Galante and to all my friends for their unending support. I would also like to thank you dear reader for taking the time to indulge this thesis work, I hope I will be able to deliver value in return.

UNIVERSITY OF TURKU
Department of Future Technologies

HUSAM HREITANI: Preserving dynamic range in fixed-point representation in 5G

Master of Science Thesis, 50 s., 4 app. p. NOKIA
January 2020

This thesis proposes a method to preserve the dynamic range of frequency samples power in the physical layer of 5G base-station by using an exponent. Keeping the resource usage to its minimum while increasing performance and preserve a high Signal-to-Noise ratio *SNR*. Operations such as addition and subtraction on samples with different exponent is made possible by implementing a common exponent block that unifies exponent with varying powers. The scope also examines multiple rounding mechanisms to keep quantization as low as possible. Unlike Block Floating-Point technique which usually operates on the multiplication stages of fast Fourier transform, Dynamic scale gives more flexibility by separating block floating-point into two parts, the first part operates on a sample level extracting individual exponents, while the second extracts the maximum exponent to execute an operations involving more than one sample. A test case implementation is proposed to give a performance comparison against full precision floating point model in Matlab.

Keywords dynamic, scale, exponent, samples

Contents

1 Introduction

This work introduces some telecommunication concepts and their corresponding mathematical basics, such as Beamforming, Fixed-Point and Floating-point representation of numbers in Chapter 2. The mathematical models and detailed functionalities of IP blocks implemented in this thesis are described in Chapter3. The Setup and RTL implementation are described in the Implementation Chapter 4. Performance testing in Chapter5 discusses the performance results of a hypothesised use case for Dynamic Scale block using Matlab, it also provides synthesis results for the RTL implementation. Finally a conclusion is drawn in Chapter 6.

2 Background

2.1 LTE and 5G

2.1.1 From LTE to $5G$

The fifth generation wireless communication ($5G$) faces many challenges to be solved and goals to be achieved, from coping with massive increasing number of mobile connected devices, rising demands for a low latency and higher connection speed, and more battery life for low power devices, while there is only limited increase in frequency spectrum from its predecessor Long term evolution (LTE) or fourth generation wireless communication $4G$. Albeit there are many incremental and new technologies implemented/to be implemented in $5G$, to understand where the scope of this thesis operates in $5G$, it is important to know some of the key concepts in both $5G$ and its predecessor $4G$. These common key concepts are Up-link, Down-link, massive MIMO and Beamforming. Massive MIMO and Beamforming are covered in more details in the next sections, while the following scenario gives an idea on what Up-link, Down-link, massive MIMO and Beamforming are, and where are they used. When a base-Station, also known as Next Generation Node B (*gNodeB*) or (*gNB*) sends data to two user equipment (*UEs*) residing in different locations in line of sight, this is called Down-Link path (*DL*), the gNB can direct the carrier signal that contains data towards the targeted user, simultaneously it can forward the corresponding data for the second user using the same frequency of carrier signal. If the two UEs started broadcasting data to communicate with the gNB, the gNB

will start receiving data to be processed and uploaded to servers simultaneously, this is called the Up-Link path (*UL*). If the two users simultaneously used the same frequency band to transmit their data to the same gNB, gNB can still manage to listen to both UEs and separate their data respectively. This spatial multiplexing is achieved using combination of many antennas forming a large array i.e. *massive MIMO* and Beamforming that exploits properties of that antenna array. Beamforming is an inherited technology from LTE, while massive MIMO is an upgraded technology from Multi-User MIMO of LTE.

2.2 Massive MIMO (*mMIMO*)

Massive MIMO is an up scaled version of Multi-User MIMO in LTE which serves multiple users simultaneously, it increases the data transfer rate by transmitting simultaneous data streams from plenty of antenna elements. Massive MIMO has an increased spatial multiplexing by performing relatively small signal processing for each antenna.[1] While Multi-User MIMO in $4G$ is limited to a maximum of 8 large high powered antennas, $5G$ mMIMO has practically dozens or hundreds of low powered small antennas. This gives many advantages for mMIMO some of which are:

1. Reliability, increasing the number of antennas will increase the capacity of Base-Station to handle more user elements, which also gives more robustness as there can be multiple spatial stream to serve the same user element.[2]

2. Better spatial multiplexing, When using more antennas more spatial streams are available thus more throughput, this gives higher spectral efficiency which is up to 10 times more efficient than multi-user MIMO in LTE.[3][4]

3. Energy efficiency, by superposition combining of transmitted signals from each antenna, the total transmitted power is inversely proportional to the number of used antennas.[5] Since each antenna in $5G$ uses less power than in LTE [6], the overall

energy usage is drastically reduced, another point is the reduced radiated power and increased throughput.

4. Security, signal jamming becomes harder due to multiple spatial streams where each antenna has a distinct propagation path.[7]

5. Reduced Cost, Antennas in mMIMO have cheaper amplifiers that uses power in range of milliwatts.[8]

6. Signal processing, using more antennas will give less interference effects and fast fading, so signal processing is simpler. [9]

7. Scalability: increasing the number of antennas in the Radio unit antenna array will enhance the performance and beams power. [2]

2.3 Beamforming

Beamforming is used to utilise the benefits of antennas in mMIMO. Similarly to a traffic signal system, Beamforming directs the beam of data transmitted from a cellular base station gNB towards the most efficient data delivery route for a UE. Another trait of beamforming is the reduced interference by avoiding the transmission into undesired directions. Beamforming helps the array of antennas in mMIMO to use the spectrum around them more efficiently. The main challenge facing mMIMO is the reduction of interference to other UEs while transmitting more information to the targeted UE using more antennas at once. Signal processing algorithms including channel estimation are used at the gNB to plot the best transmission route to reach each UE. This route may even include bouncing the carrier signal off walls and other objects in a coordinated and precise manner. In the UL path, the angle in which information is received from each UE is computed by computing the delay time in which the signal hits neighbouring antennas in the antenna array, since the distance between each antenna in the array is known. This will allow the Radio

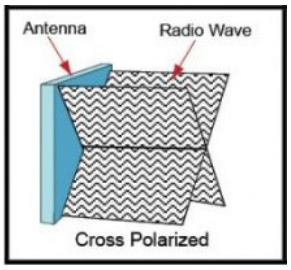


Figure 2.1: Cross polarised antenna

unit to super position the signals received at each antenna by using phase shifting to amplify the received signal for each UE while cancelling out simultaneous signals received from the rest of UEs. In order to form the beam in which each antenna in the array will beam, a matrix of beamforming weights is constructed, then multiplied by each stream of data inside the Radio unit. One complex beamforming weight for each antenna will generate the signal of each individual antenna. A beam of data transmitted or received from a certain space is called spatial stream or (*SxC*). A stream of data coming from each antenna in the antenna array is called transceiver stream or (*TRx*. If for example a radio unit has antenna array of size 4x2x2 where 4x2 is the number of array elements while each antenna has 2 polarities referred as cross-polarised, then the total number of TRxs will be 16. A cross-polarised antenna means there are two planes from which the electric field component of the EM radiation is oscillating, a cross-section of these two planes looking at the antenna will give the shape of letter (X) as shown in figure 2.1.

The basic concept that beamforming relies heavily on is super positioning of signals, for example, if two sinusoidal waves with the same amplitude and frequency are sent or received from two adjacent antennas as shown in figure 2.2, the combined resulting signal will be four times (x4) the power of one of the component signals, That is due to $P = \frac{U^2}{R}$. In contrast when adding those two signals with a phase difference of 180° the resulting signal will be zero and the two components will completely cancel out each other as in figure 2.3. The power gain in overlapping received or sent signals is called Antenna gain which is the result of signals being added in super positioning. [10] By beaming the

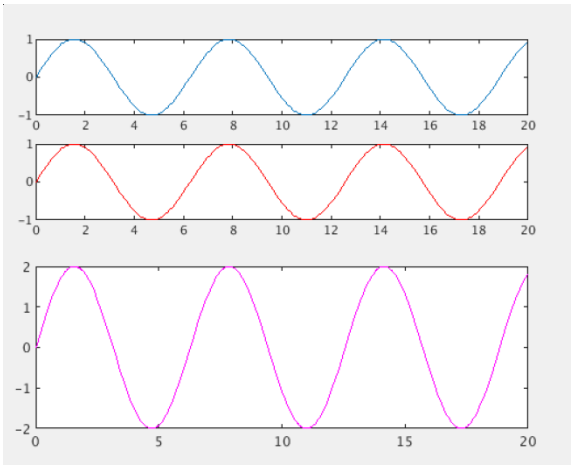


Figure 2.2: Superposition Addition of two sinusoidal waves.

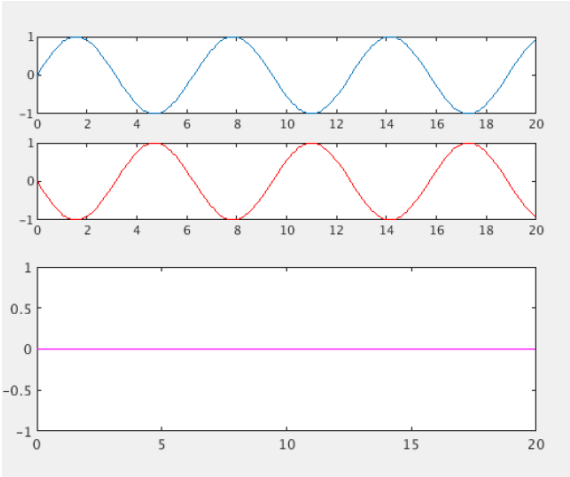


Figure 2.3: Addition of two sinusoidal waves with 180° phase difference.

same signal from the antenna array using its many small antennas that are placed apart by a small known distance in a way that will overlap the transmitted signals travelling in a certain direction and combine their power 2.4, the resulting signal transmitted in that direction will have a power sum of all the signals beamed in that direction plus a gain power from the antennas forming the beam. In summary the more overlapping signals in a constructive interference direction the more gain will be achieved.

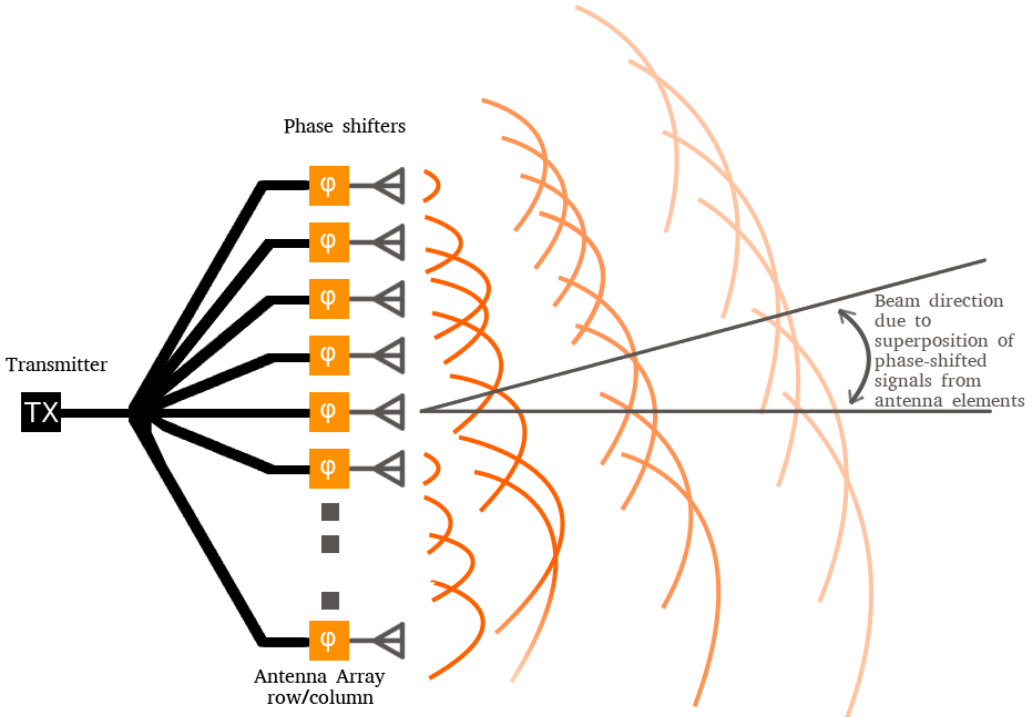


Figure 2.4: Phased array antenna setup

2.4 IQ samples

In Up-Link path (UL) data is received as time domain samples which are sampled from antennas using a sampler. The time domain samples containing the amplitude and angular frequency of the received signal are represented in the complex plane as Cartesian coordinates *IQ Sample*. The in-phase (*I*) is represented on the Real axis coordinate and the quadrature *Q* is represents on the Imaginary axis. Components are called in-phase and quadrature because they represent the received signal sin(x + ϕ) using two orthogonal sinusoid components sin(x) cos(ϕ) + sin(x+$\frac{\pi}{2}$) sin(ϕ), having x = $2\pi ft$, where f is the frequency and t represents time, while ϕ is the signal phase. in both components, cos(ϕ) and sin(ϕ) are called carrier signals. The first component sin(x) has the same frequency hence it is in-phase with the represented signal, whereas the second component sin(x+$\frac{\pi}{2}$) has $\frac{\pi}{2}$ or quarter of a cycle difference, hence the quadrature.[11]

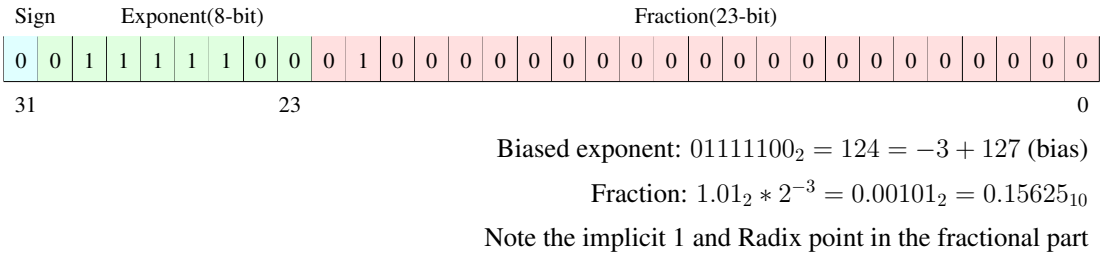
Received time domain signal is passed to a Fast Fourier Transform block (*FFT*) that gives out Frequency domain samples, these samples are also represented in *IQ sample* format in the complex plane. Frequency samples represents data of users carried over many orthogonal sub-carriers, this is known as Orthogonal frequency division multiple access (*OFDMA*). In a real industry hardware application, IQ samples are represented in either fixed-point (FXP) or floating-point (FLP) format.

2.4.1 Fixed- and Floating-point representations

Digital Signal Processors (*DSPs*) are categorised into Fixed-point number and Floating-point number DSPs. The difference between the two representations is related to the Radix point. In binary terms, a floating-point number consists of a mantissa part and an exponent part, the amount of bits for mantissa defines the accuracy a number can represent, and the number of exponent bits defines the range a number can have. In decimal terms a floating-point number is able to have a Radix point that can "float" representing a

Figure 2.5: Example of representing 3.5 in two different Fixed-point formats

Figure 2.6: Single precision IEEE 754 floating-point Example



wide dynamic range between represented numbers, for example 1.7654321 and 1234567.5

In fixed-point the Radix point position has a predefined fixed place, a represented number contains certain amount of bits for representing its integer part and another for its fractional part. Figure 2.5 illustrates how to represent decimal 3.5 in two different 6 bits fixed-point formats, first format is 1.3.2 (3 integer, and 2 fractional bits) and the second format being 1.2.3 (2 integer, and 3 fractional bits). A fixed-point representation has limited dynamic range of numbers that it can represent, however it provides better accuracy than floating point given a fixed word length numbers.

Some standards for floating point representation includes the IEEE Standard 754 which is an industry standard, it has multiple levels of precision, the commonly used two are single precision 32 bits and double precision 64 bits. A single precision can represent $\pm 1.18 * 10^{-38}$ to $\pm 3.4 * 10^{38}$. The structure of a single precision floating point number can be found in figure 2.6 [12]

Floating-point numbers stored and operated on represent numbers with a trade off between precision and range. In floating-point representation the Radix point can "float"

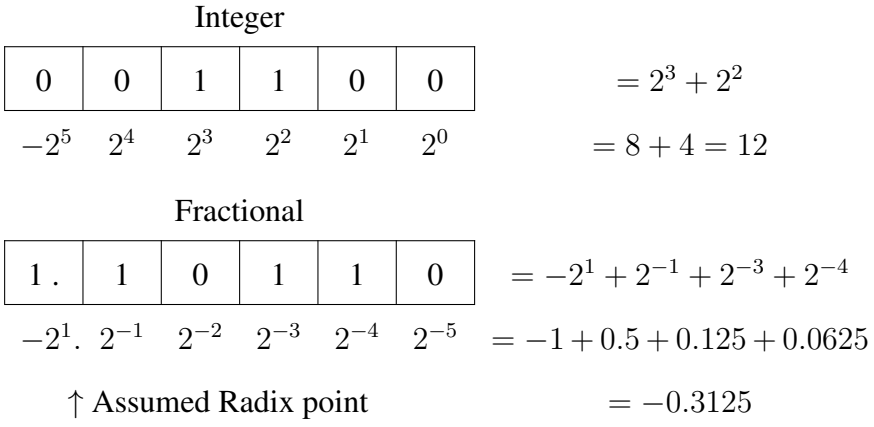


Figure 2.7: Example of Fixed-point representation for integer and fractional numbers

between digits of a represented number, giving more bits to represent the fractional part, or the integer part depending on the size of the number. For example, representing the number 0.00005 can be done by $5 * 10^{-5}$ while the number 500000 is done as $5 * 10^5$, So with the use of exponent to indicate the Radix point position, a wide range is represented using less digits.

Multiplication in Fixed-point format

When multiplying two numbers in a Digital signal processor (DSP) using fixed-point format, the resulting number has a width equals to the sum of both multiplied numbers. In a defined width DSP, it is necessary to represent numbers in operations using fractions i.e. $< |1|$ to give mathematically correct results because only accuracy will be lost when LSB bits are dropped. If numbers are not representing fractions and instead representing integers, then dropping LSB bits will change the actual value of that number, thus giving wrong results. Results of addition and multiplication DSP operations will have more bits (bit growth) than their respective input, to represent these results correctly and perform further DSP operations in the same FXP format without overflow, LSB bits must be dropped. The integer part width in a result of multiplying two numbers equals to the

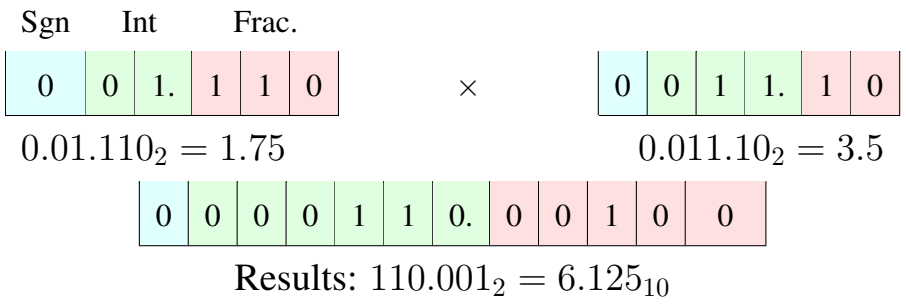


Figure 2.8: Different Fixed-point formats multiplication

sum of integer parts of its components +1 bit which is a sign extended bit. The fractional part of resulting number equals to the sum of its component fractional parts. For example, when multiplying two number having the format 1.2.3 * 1.3.2 the resulting number will have the format 1.6.5 as illustrated in figure 2.8.

2.5 FFT operation in UL

IQ samples at FFT output represents a frequency signal that consists of sub-carriers, the phase and amplitude of each sub-carrier has been already modulated by the user's data received by gNB. This data is extracted from each sub-carrier in the demodulation phase. In $5G$ multiple users can share the same sub-carrier frequency if they reside in different spatial location, this is achieved with beamforming as discussed earlier, in later processing stages data is separated for each corresponding user. For simplicity and without loss of generality, cases involving multiple users sharing the same frequency bandwidth will not be considered in this thesis scope.

2.5.1 FFT

Mathematics of FFT

Fast Fourier Transform (*FFT*) is an algorithm for computing Discrete Fourier Transformation (*DFT*) that is defined, for N-point transformation, by the following equation:

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{nk}$$

Where $X(k)$ is frequency bin number k and the twiddle factors are:

$$W_N^{nk} = e^{-j\frac{2\pi}{N}nk}$$

Cooley–Tukey FFT algorithm breaks down the previous equation into butterfly stages (addition and subtraction) and a twiddle factor multiplication as shown in figure 2.9. $log_2 N$ is the total number of stages for a radix-2 FFT size of N. Even frequency bins and odd frequency bins are computed in the following equations.

$$X(2k) = \sum_{n=0}^{\frac{N}{2}-1} [x(n) + x(n + \frac{N}{2})] W_N^{n2k}, k = 0, 1, 2, .., \frac{N}{2} - 1$$

And:

$$X(2k+1) = \sum_{n=0}^{\frac{N}{2}-1} [x(n) - x(n + \frac{N}{2})] W_N^{n(2k+1)}, k = 0, 1, 2, .., \frac{N}{2} - 1$$

Figure 2.10 illustrates the amplitude of a complex signal $e^{i\theta * t}$ having frequency $\theta = 50Hz$ is translated into frequency spectrum with maximum amplitude of $N = 4096$ for the real part.

Power in FFT

When computing a frequency bin, multiplying with twiddle factors then summing at the butterfly stage may result in high amplitude for that frequency, this is due to signals in time domain input having the same or close frequency to that frequency bin. Since an

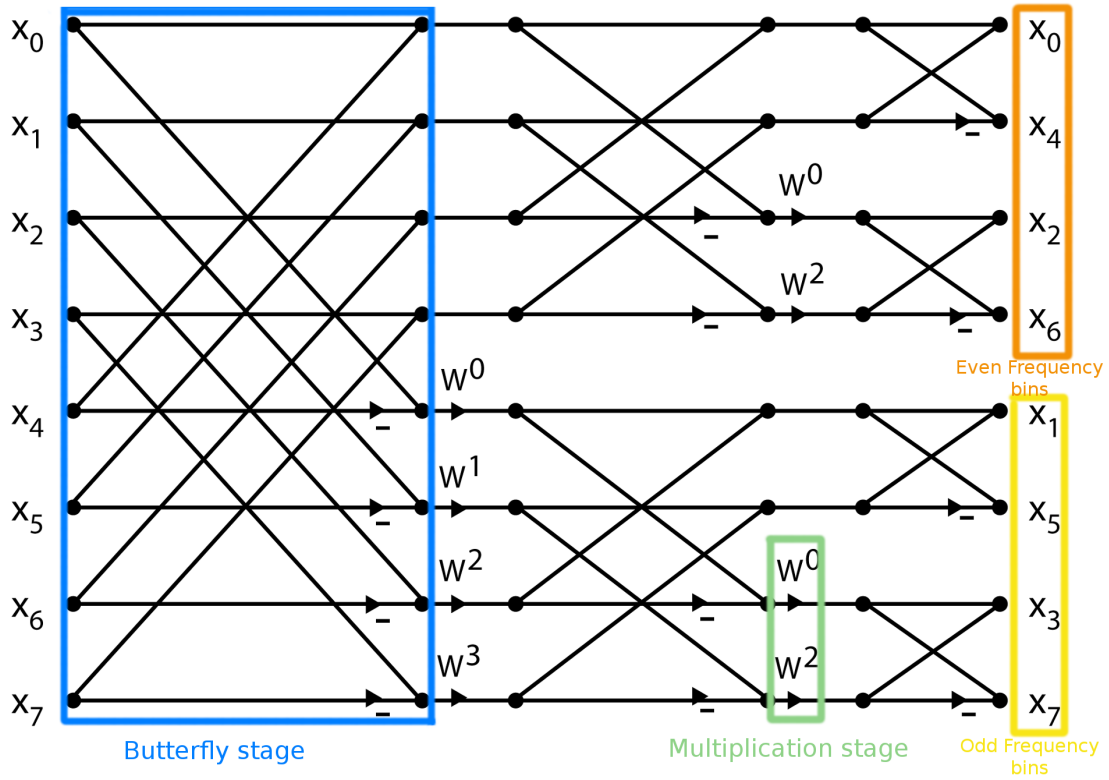


Figure 2.9: Cooley-Tukey FFT stages

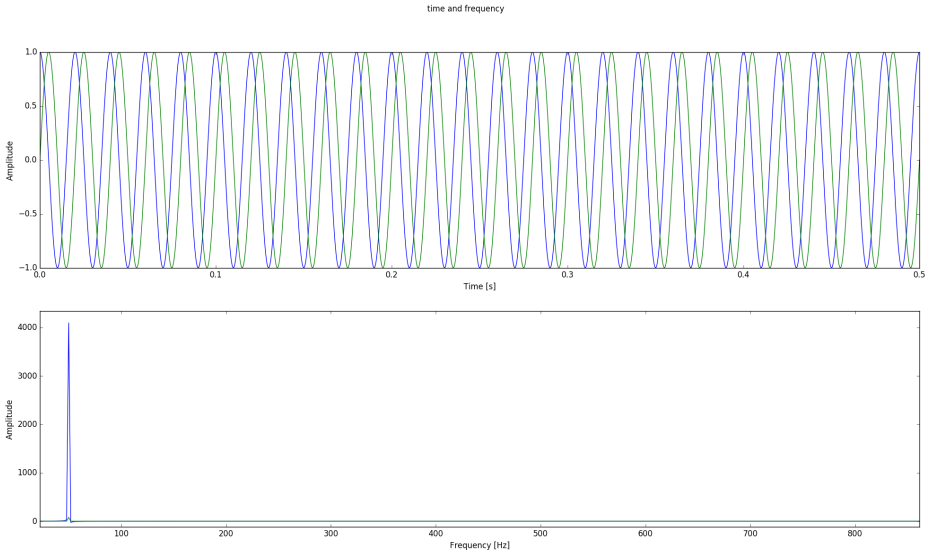


Figure 2.10: Time and Frequency domain complex signal

FFT is only a different representation of the same signal, the energy of both FFT input and output should be the same. A normalisation process on FFT output is required to that. Energy of time signal is represented by the left hand side of the following equation, while energy of frequency bins is represented by the right hand side, this is also known as Parseval's theorem:

$$\sum_{n=0}^{N-1} \left|x[n]\right|^2 = \frac{1}{N} \sum_{k=0}^{N-1} \left|X[k]\right|^2$$

OR

$$\sum_{n=0}^{N-1} \left|x[n]\right|^2 = \sum_{k=0}^{N-1} \left|\frac{1}{\sqrt{N}} X[k]\right|^2$$

Thus normalising the resulting IQ samples at the FFT output is done by dividing each frequency sample by $\sqrt{N}$ where N is the FFT size. [13]

Bit growth in FFT

Assuming that each time domain sample entered to FFT IP block is represented in fixed-point, with format 1.0.15 for both I and Q parts. Frequency domain IQ samples at the FFT IP block output have the format: $1.L.15$ where the first MSB bit corresponds to the sign bit, the 15 LSB corresponds to the fractional bits and L are integer bits corresponding to maximum bit growth, which is dependant on FFT window size. In figure 2.10 that size is 4096, which needs 13 bits to represent in binary. The bit growth of FFT butterfly and multiplication stages can be formulated into the following equation: $L = Log_2(FFT\_SIZE) + 1$ The supported FFT window sizes done in this thesis are of power 2 (radix-2) and can have one of the following values: $[256, 512, 1024, 2048, 4096]$. Accordingly the following FFT size values $[256, 512, 1024, 2048, 4096]$ will correspond to a bit growth part in the output IQ samples of width $[9, 10, 11, 12and13]$ respectively. After normalising the IQ samples by dividing them over $\sqrt{FFT\_SIZE}$, their format becomes $1.K.15$, K being the new integer bit growth width that is derived from the following

bit growth			
FFT_SIZE	L	$\lfloor Log_2(\sqrt{FFT\_SIZE}) \rfloor$	K
4096	13	6	7
2048	12	5	7
1024	11	5	6
512	10	4	6
256	9	4	5

Table 2.1: Value of bit growth after FFT Normalisation

equation: $K = L - \lfloor Log_2(\sqrt{FFT\_SIZE}) \rfloor$ Consequently K will have the corresponding values shown in table 2.1.

3 IP blocks Mathematics

In this section the mathematical functionality of implemented IP blocks are discussed in details. The blocks are first modelled in C++ to give a correct mathematical representation that is later used for verification and performance testing. The RTL code of IP blocks and test-benches are implemented using VHDL.

3.1 Basic concepts

3.1.1 Exponent value and sign

IEEE Standard 754 Floating-point representation covers a very wide range of numbers, however it uses at least 32 bits to represent a number, which can be cumbersome and computationally heavy. A fixed-point (FXP) representation requires less HW resources and computation complexity, yet it does not use an exponent at all, which limits the range of numbers it can represent to -2^N through $2^N - 1$ with N being the number of bits in the FXP representation. A hybrid solution is to keep the lightweight FXP to represent samples while using a separate exponent number to represent the power of these samples caused by the bit growth after FFT operation. A similar concept in the sense of having a common exponent for multiple numbers is implemented in DIT FFT which is called block floating-point [14].

Suppose the results of multiplying 4×4 in fixed point format $1.3.0$ needs to be stored in the same fixed point format. Ideally the results should be stored in format $1.(1+3+3).0 =$

$1.7.0$ or $0100 \times 0100 = 00010000$ which is more hardware resource demanding. To represent using the format $1.3.0$, first, the sign extended bits can be reduced since they equal to the sign bit and removing them will not change the result i.e. $00010000 = 010000$. So far the number 010000 has two excess bits to fit into the format 1.3.0, to resolve this issue an exponent of 2 is introduced 010, and the multiplication results is stored as 0100 To reproduce the original number, the results is left shifted by the value of exponent i.e. $0100 << 010 = 010000$.

As results only grow in size, exponent will always be positive, thus it can be represented as unsigned in the format $0.3.0$.

3.1.2 Down scaling and Rounding

When scaling down numbers due to design constraints on how many bits can be used to represent numbers quantization noise increases. In order to reduce the level of quantization noise, down scaled numbers are also rounded. There are multiple rounding mechanisms implemented in this thesis and different options have different level of quantization noise on the overall set of scaled down numbers, albeit a very effect.

1. Directed rounding: x is the rounded number

   (a) Ceiling $\lceil x \rceil$, i.e. the largest integer not exceeding x. This is achieved in binary form by adding 1 to the number after truncation if the truncated fraction is not equal to 0. Example of a positive rounding from fraction to integer:

   $3.0625 \approx 4$

   $0011.0001_2 = 3 + 0.0625 = 3.0625$

   $0.0625 \neq 0$

   $0011.0001_2 >> 4 + 1 = 0100_2 = 4$

   The same applies to negative numbers: $-3.9375 \approx -3$

   $1100.0001_2 = -8 + 4 + 0.0625 = -3.0625$

$0.0625 \neq 0$

$1100.0001_2 >> 4 + 1 = 1101_2 = -8 + 4 + 1 = -3$

This is equivalent to adding $2^{\#Truncated\_Bits}$ Ceiling introduces large bias towards positive infinity.

(b) Floor $\lfloor x \rfloor$, i.e. the smallest integer not exceeding x, This is achieved by simple truncation to reach the desired length. An example for a positive rounding of fractional number to an integer is:

$3.9375 \approx 3$

$011.1111_2 = 3 + 0.5 + 0.25 + 0.125 + 0.0625 = 3.9375$

$011.1101_2 = 011.0110 >> 4 = 3$

similarly for negative numbers:

$-3.0625 \approx -4$

$1100.1111_2 = -8 + 4 + 0.5 + 0.25 + 0.125 + 0.0625 = -3.0625$

$1100.1111_2 >> 4 = 1100_2 = -8 + 4 = -4$

Floor introduces large bias towards negative infinity.

(c) Round to infinity $sign(x)\lceil |x| \rceil$ This achieved in binary numbers by adding 1 to the truncated results in case of positive numbers i.e. ceiling, In case of negative number only by truncation is applied. Round to infinity is symmetrical when rounded numbers are uniformly distributed between negative and positive.

(d) Round to zero $sign(x)\lfloor |x| \rfloor$ Contrast to round to infinity, Round to zero will truncate positive numbers, and add 1 to truncated negative numbers. Round to zero is also symmetrical in case of uniformly distributed rounded numbers between negative and positive signs. All four previous rounding methods introduce large quantization error.

2. Rounding to nearest: has less quantization error than Directed rounding. If a fractional number is less than exactly half then it is rounded towards the smaller nearest number, if that number is greater than half the amount, it will be rounded towards

the largest nearest number. However when the number to be rounded resides exactly between the two numbers that can be represented, then one of the following methods is applied:

(a) Round half up, This will round numbers residing exactly in the middle towards positive infinity. In binary terms this is achieved by adding half of the smallest number that can be represented, in the following example the resulting rounded numbers are integers thus half (0.5) is added.

$3.5 \approx 4$

$0011.1000_2 = 2 + 1 + 0.5 = 3.5$

$0011.1000_2 + 0.1000_2 = 0100.0000_2 >> 4 = 0100_2 = 4$

And in case of negative numbers:

$-3.5 \approx -4$

$1100.1000_2 = -8 + 4 + 0.5 = -3.5$

$1100.1000_2 + 0.1000_2 = 1101.0000_2 >> 4 = 1101_2 = -3$

This rounding mechanism introduces a small bias towards positive infinity, but is the cheapest in terms of resource usage.

(b) Round half down, Conversely from Round half up, this method will round numbers falling exactly in the middle of two possibly represented numbers (in this example integers 3 and 4) towards negative infinity. A simple check to see if the number to be rounded falls exactly in the middle is by adding 1's to all digits that will be truncated except the MSB bit as follows: $3.5 \approx 3$

$0011.1000_2 = 2 + 1 + 0.5 = 3.5$

$2^{(4-1)} - 1 = 0.1000_2 - 1 = 0.0111 = 0.4375$

$0011.1000_2 + 0.0111_2 = 0011.1111_2 >> 4 = 0011_2 = 3$

similarly for negative numbers: $-3.5 \approx -4$

$1100.1000_2 = -8 + 4 + 0.5 = -3.5$

$1100.1000_2 + 0.0111_2 = 1100.1111_2 >> 4 = 1100_2 = -4$

This method requires a little bit more resource than the previous one, and introduces small bias towards negative infinity.

(c) Round half even, Numbers falling exactly at the middle are rounded towards the nearest even number, a binary number is even when its integer part ends with 0 and it is positive, or 1 in case it is negative. As a result, rounding an even number will be toward zero, and an odd number will round it towards infinity (negative infinity if it is negative number). For example, 5.5 is rounded up to 6, while 4.5 is rounded down to 4. To find if the number to be rounded is even or odd, the last bit before truncation chunk is checked, if it equals to 1 then half is added, and the number is truncated. If the last bit before truncated chunk equals to 0 then $0.1 - 1 = 0.0111$ is added to the number then truncation is done. This will ensure that the number resides exactly in the middle. Adding the number of fractional bits -1 e.g. 0.0111 will have no effect on the number if it resides less than or exactly in the middle (half), but if it is greater than half, the addition will round it to the next integer. An example of positive even round truncating the last 4 bits:

$4.5 \approx 4 : 0100.1000_2 \approx 0100_2$

The 5^{th} LSB bit equals to 0, then $0.1000_2 - 1 = 0.0111_2$ is added: $0100.1000_2 + 0.0111_2 = 0100.1111 >> 4 = 0100_2$. The last 4 bits are truncated which results in $0100_2 = 4$. Another example of positive odd number i.e. integer part ends with 1, then it will be rounded up to the nearest even integer:

$3.5 = 0011.1000$ is rounded to 4 by adding half 0.5 or 0.1000 the result is then truncated: $0011.1000_2 + 0.1000_2 = 0100.0000_2 >> 4 = 0100_2 = 4$.

If the number is negative and its integer part ends with 1 then it is an even negative number, and the way it is rounded is exactly the same as positive odd number where the integer part ends with 1. if it ends with exactly half then it is rounded up by adding 0.5, e.g.

$-2.5 = 1101.1000_2 = 1101.1000_2 + 0.1000 >> 4 = 1110_2 = -2$

In case the integer part is negative and odd (ends with 0), then it is rounded down by truncation similarly to a positive odd number. $-3.5 = 1100.1000_2 = 1100.1000_2 + 0.0111 = 1100.1111 >> 4 = 1100_2 = -4$.

Round half even method gives a more symmetrical rounding than the previous two methods since it rounds numbers exactly in the middle up and down depending on its sign and if it is even or odd, However it is more resource expensive than the previous two methods.

(d) Round half odd, Contrary to round half even, round half odd rounds number residing exactly in the middle of two numbers to the nearest odd one. So even numbers are rounded to infinity, while odd numbers are rounded towards zero. for example: 4.5 is rounded to 5, while 3.5 is rounded to 3. -4.5 is rounded to -5 and -3.5 is rounded to -3. if number is positive and the last integer bit is 0 then add 0.1000 or half then truncate: $4.5 = 0100.1000_2 + 0.1000_2 = 0101.0000_2 >> 4 = 5$. When the number is positive and odd then:

$3.5 = 0011.1000_2 + (2^{(4-1)} - 1)$

$0011.1000_2 + 0.1000_2 - 1 = 0011.1000 + 0.0111$

$0011.1111 >> 4 = 0011_2 = 3$.

If number is negative and its integer part ends with 1 then it's treated the same way as a positive integer ending with 0. e.g. $-3.5 = 1101.1000_2 = 1101.1000_2 + 0.0111 = 1101.1111 >> 4 = 1101_2 = -3$

Similarly to rounding a positive integer ending with 0, a negative number with its integer part ending with 1 (even) is rounded by adding half then truncating. e.g. $-4.5 = 1011.1000_2 + 2^{(4-1)} = 1100.0000_2 >> 4 = -4$

3.2 Dynamic scale

The main idea behind using the exponent for IQ samples in Dynamic Scale is to preserve the dynamic range power for IQ samples. When a high power difference between received samples exists, then samples with high power can have an exponent value to indicate their power level, while samples with low power are kept and not dropped. Preserving the dynamic range of FFT output samples could be a potential use case for the Dynamic Scale block. Because the received power of different UEs (transmitting in different frequencies) can vary significantly. Consider performing an FFT operation on a time domain signal that consists of three sine waves with an overall peak-to-peak amplitude of 2 such as the one depicted on the left side of figure 3.1 will give a frequency output response as depicted on the bottom side of figure 3.1. The absolute value of frequency bin 50Hz is equal to 57.5994, while frequency bins 100Hz and 200Hz are equal to 3.1867 and 3.1914 respectively, since numbers are represented in fractions within boundary of]1, -1], the value of FFT output should be normalised by the largest possible value which is 64, thus the corresponding frequency bin values will be $\frac{57}{64}$, $\frac{3.1867}{64}$ and $\frac{3.1914}{64}$, or 0.89998, 0.04979 and 0.04986 respectively, this equates to 0.111001100110010, 0.000011001011111 and 0.000011001100001 respectively in binary fixed-point format with 15 bits fractional part. Assuming that a hardware setup only supports 5 digits of fractional part, this will results in the large frequency bin of 50Hz to become 0.11100 or 0.875 and the frequency bins of 100Hz and 200Hz to be dropped to 0.00001 or 0.0313. This amounts to 2.77% loss for frequency bin 50Hz and about 37.13% loss for the other two signals which is relatively a high loss. If Dynamic Scale IP is used then the results will be 0.11001 with a corresponding exponent of 2 (computation will be explained later). 0.11001 is then scaled down to 0.000011001 or 0.04736 by multiplying with it's exponent 2^2 and dividing by 64. giving a loss of 4.87% of the original signal. This example has an intentional bad design choice used only to illustrate the functionality of exponent, in reality HW bit width is chosen to accommodate 15 bits for the fractional part, which makes much less SNR loss even

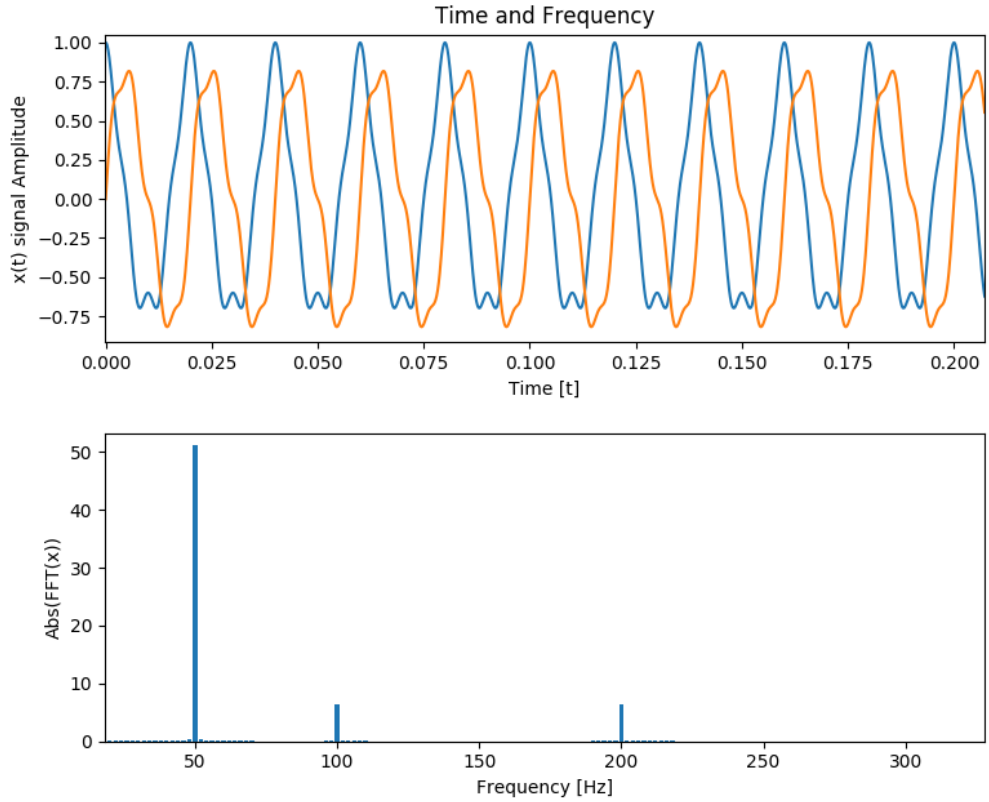


Figure 3.1: Time and Frequency domain complex signal

without using the exponent.

3.2.1 Design constraints

Some design decisions taken in designing this IP block are presented here.

Exponent value in Dynamic Scale

The number of bits to represent IQ samples in this design is chosen to be 32 bits where 16 bits are used to represent the imaginary part (Q) and 16 bits to represent the real part (I), both I and Q have the format $1.0.15$. An exponent is also introduced in order to represent IQ samples of form 2x $1.K.15$ as 2x $1.0.15$. Since K can have a maximum value of 7 bits when the maximum FFT size is 4096, the exponent used to represent K is 3 bit wide, i.e.

the exponent indicates how many right shifts were applied to $1.K.15$ to make it $1.0.15$.

Down scaling and Rounding in Dynamic Scale

IQ samples provided at the input of Dynamic scale block are 29+29 bits wide, while at the output the bit width is 16+16. This calls for scaling down at multiple stages. To reduce the quantization error while scaling down, rounding is used whenever down scaling is applied. The two stages in which scaling and rounding down happens are:

1. pre-multiplication rounding
2. post-multiplication rounding

The multipliers provided by hardware used to normalise I and Q sample have 27 bits wide input. Those multipliers are depicted in figure 3.2 as an encircled x which depict a multiplication in a DSP. I and Q parts of the sample should be scaled down and rounded from 29 to 27 bits and from 28 to 27 bits in cases of 4096 and 2048 FFT_SIZE respectively. At the output of the DSP multiplication another rounding takes place where samples are scaled down from 1.15.38 to 1.0.15 bits, DSP output gives numbers with implicit format of 1.15.38 as explained in 2.4.1. For both rounding stages round half up to nearest mechanism is chosen, this is because it has the least resource usage while still providing low quantization noise results.

3.3 Common Exponent Scale

The main idea of Common exponent is to unify the exponent power of multiple samples received at its input in parallel, this is necessary to ensure the correctness of mathematical operations such as addition and subtraction on these samples. Common exponent scale will choose the maximum exponent between all input samples and scale the reset of the samples according to the difference between their exponent and the maximum found.

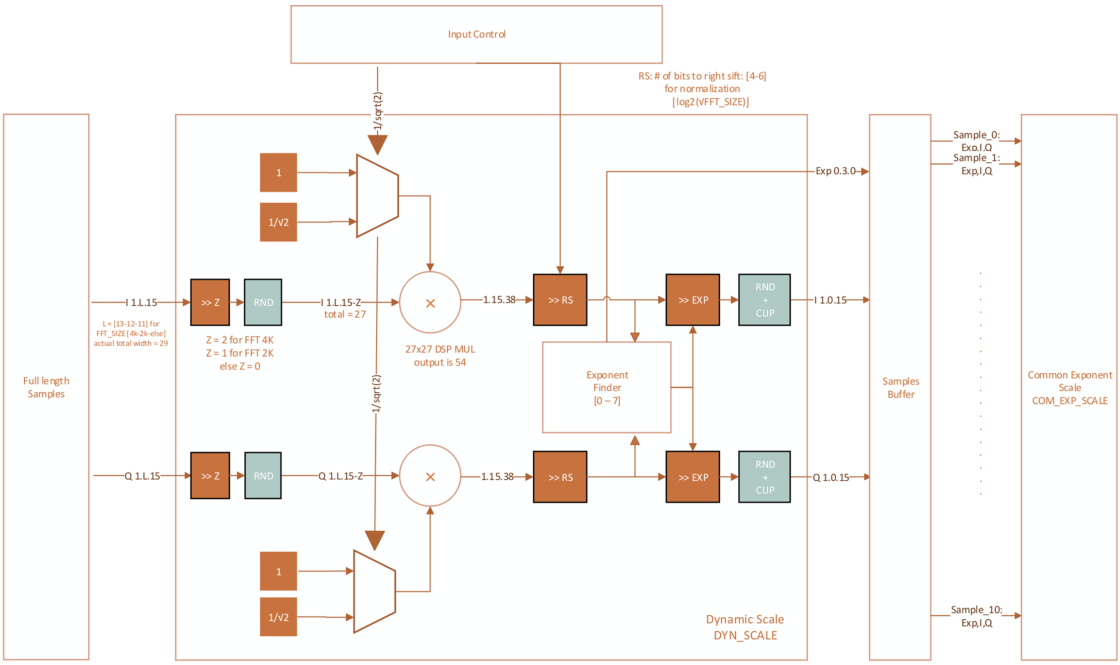


Figure 3.2: Dynamic scale block diagram

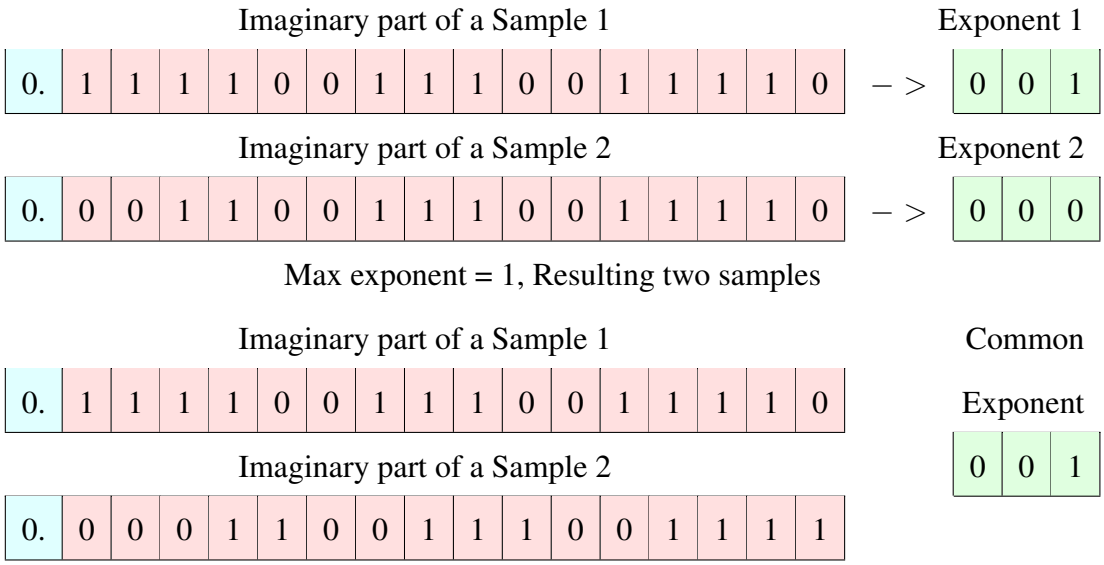


Figure 3.3: Example of common exponent, Sample2 is right shifted by 1 to match the exponent power.

When samples are scaled according to the maximum common exponent, samples with lower exponent will loose some accuracy due to scaling down (right shift) to increase their exponent. Down scaling and rounding lower exponent samples is necessary to match their exponent with the maximum exponent found. Samples with maximum exponents have the sign extended bits already reduced in Dynamic scale, i.e. they are saturated, so they cannot be left shifted to reduce their exponent power, and only samples with lower exponent power can be right shifted instead to reach the same power level. Figure 3.3 illustrates an example of how two random imaginary parts of complex samples with different exponents are scaled. sample 2 has an exponent of zero while sample 1 has exponent one, so the first sample is scaled down by one bit which is the difference between the two exponents to match the exponent of sample 1. Common Exponent Scale also rounds the down scaled samples using the same half up to nearest mechanism Dynamic scale block uses.

Because the exponent represent the power of a sample, it is common between the imaginary part and the real part of that sample. When Common exponent scales down

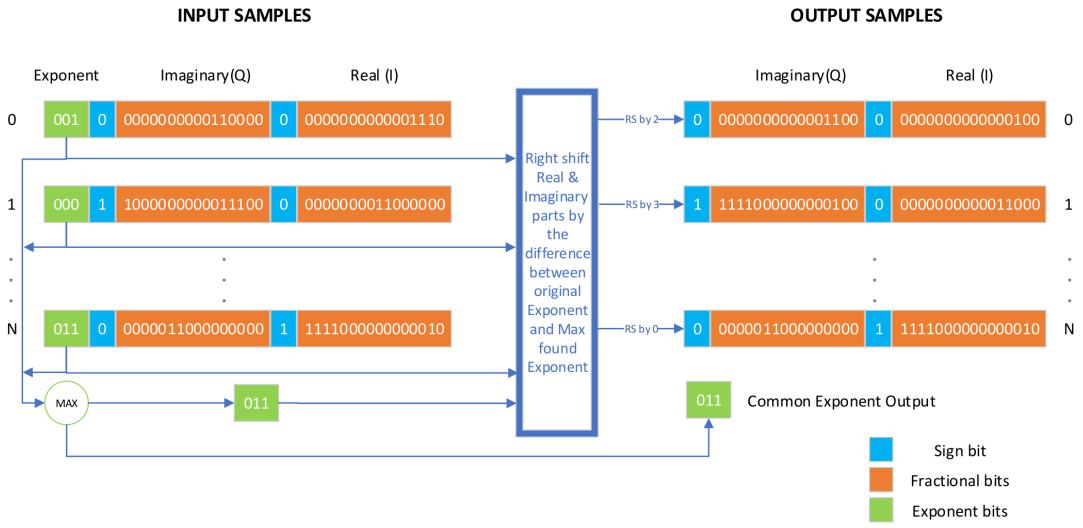


Figure 3.4: Common Exponent Scale functionality

the sample to increase the value of its exponent, both the imaginary and the real parts are scaled of the sample are scaled by the same amount. Figure 3.4 illustrates how multiple samples are provided to the input of Common exponent scale block, each sample has its exponent at the MSB 3 bits, then the imaginary part occupying 16 bits, then the real part in the 16 LSB bits. After comparing exponents of all provided samples, a maximum exponent of 3 or 011_2 is found, and each sample is scaled by the difference as mentioned before. Notice how both the imaginary part and the real part are scaled by the same amount. Notice how the rounding is used in both the positive real part of the first sample, and the negative imaginary part of the second sample. 0.000000000001110->0.00000000000100 And 1.100000000011100->1.11100000000100.

4 Setup and Implementation

4.1 Hardware Setup

In this section the platform in which the project is implemented on is described. There are many commercial FPGAs and development boards available on the market nowadays. Altera stratix family provides a comprehensive industrial benchmarking to test the performance and functionality of these IP blocks. The IP blocks are developed and tested using Intel Quartus, and the Altera Stratix 10 board is chosen to compile the design. Stratix 10 board has multiple types of DSP multiplication blocks the block used is the 27x27 input bits wide block that has output of 54 bits wide. [15] The FPGA chosen in this project is the Startix 10 SX 2800 FPGA [15], this choice is arbitrary and can be replaced by any other FPGA that can satisfy the basic resource usage requirement. Although the Stratix 10 SX has a maximum processor frequency of 1.5 GHz, the actual fabric runs at a lower speed, usually around 300 MHz. Thus the maximum fabric frequency chosen for synthesis is 500 MHz in which the implemented IP blocks can operate in under a maximum temperature of 100 degrees Celsius. Intel utilises FPGAs using Adaptive Logic Modules or *ALM* which is the basic resource usage block for optimized performance.[16] Quartus synthesises a function with up to eight inputs and eight outputs as a single ALM. Each ALM contains two or four register logic cells, two combinational logic cells, and two dedicated full adders, a register chain, a carry chain and a 64-bit Lookup Table *LUT* mask. An ALM can operate in arithmetic mode, extended LUT mode, combinational logic mode,

or shared arithmetic mode. A shared arithmetic mode is when two functions share some common inputs and the total of their combined inputs is equal or less to eight, in such case Quartus may synthesis those functions in one ALM. In this particular FPGA model, there are 933,120 ALMs, which corresponds to 3,723,480 ALM registers. The total number of Variable-precision digital Signal Processing (DSP) blocks in the Stratix 10 SX 2800 is equal to 5,760 DSPs.

4.2 Software Setup

The RTL design is implemented using VHDL based on the block diagrams and design specifications. RTL code is verified over three stages, these are

1. C reference model.
2. Sanity check.
3. Universal Verification Methodology.

Each stage is explained in this section.

4.2.1 Reference Models

A bit exact implementation of Dynamic scale and Common Exponent Scale blocks are created using C++ language and compiled using a GCC compiler on a Linux machine running Ubuntu 16.04.12. The GCC version is: 5.4.0 Each HDL process in the RTL code has a peer function in C++ that replicates the same mathematical process. The reference model is made by modelling Dynamic scale and Common Exponent Scale design functionality with a bit accurate mathematical C model, the reference model is used for comparison against the RTL results. The performance and mathematics of C models are tested using Matlab for different scenarios.

An extra functionality the C Reference Model has is that it generates random input file which is used as a stimulus for RTL test-bench and the reference model itself.

4.2.2 ModelSim

ModelSim is a tool provided by Mentor Graphics used to simulate the RTL IP blocks via their respective test-benches. The Revision of ModelSim used in this project for simulation is 2019.01. Sanity check and simple Test-benches (TB) are implemented in VHDL to test the functionality and correctness of signals going through the IP blocks. After each individual functionality of each IP block is tested, the collective functionality is tested. The Test-benches for Dynamic Scale and Common Exponent Scale are stimulated by the C reference Model file of randomly generated input. The generated input file is constrained by the corresponding Hardware bit width of the RTL implementation. Test-benches output signals value are written to an output sink file. Collected results in the sink file are compared against the output of C reference model which is fed the same input.

4.2.3 Universal Verification Methodology (UVM)

The RTL code has been formally verified by the verification team to match the expected behaviour for all the specified functionalities. Sanity check and in simple verification however is covered in the scope of this thesis.

4.2.4 Quartus

Intel Quartus Prime is the platform in which the IP blocks are synthesised on. The chosen Stratix 10 FPGA is selected as the platform in which IP blocks are synthesised on. The version of Intel Quartus Prime used for synthesising the IP blocks on a virtual Stratix 10 FPGA is 10.1.0 build 240.

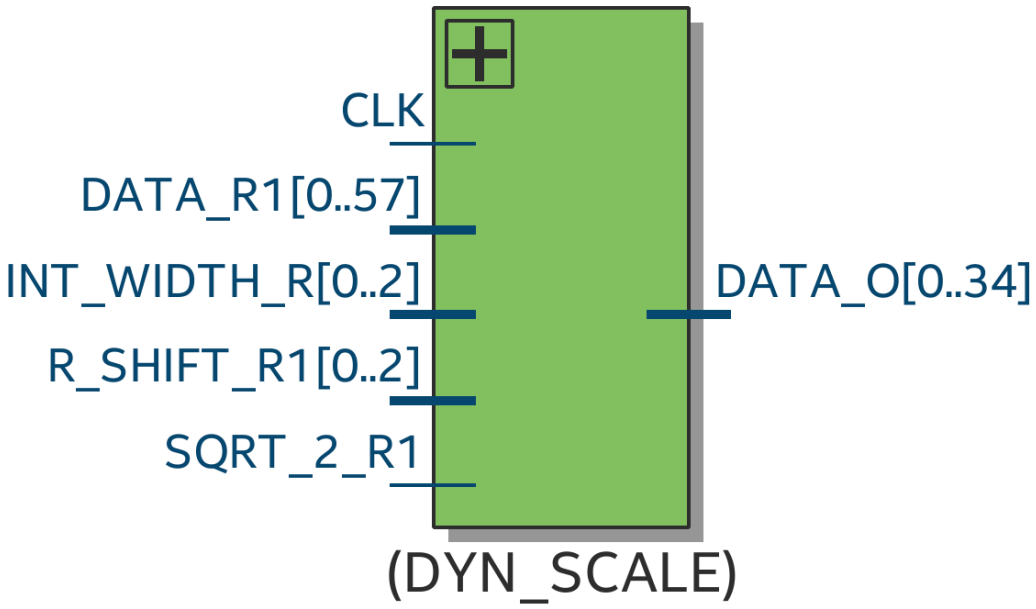


Figure 4.1: Dynamic Scale Core Ports

4.3 Implementation

4.3.1 Dynamic scale

Dynamic Scale shown in the block diagram 3.2 depicts a wrapper that contains a single Dynamic Scale core. The number of cores instantiated in the wrapper is defined in a generic that corresponds to the FFT engine architecture requirements. IO ports of a single Dynamic Scale core are depicted in figure 4.1. A brief functionality of ports of each core are:

1. CLK : input logic bit for clock signal.
2. SQRT_2_I: input control bit used to define the multiplication factor.
3. R_SHIFT_I: input number of bits to right shift in normalisation process.
4. INT_WIDTH_I: input specifying the number of integer bits in IQ samples.
5. DATA_R: input data of I and Q parts of the sample.

6. DATA_O: Output data of IQ sample and its extracted exponent.

Each Dynamic Scale core can be divided into sub-blocks that performs mathematical functionality, these sub-blocks are:

1. Pre-Multiplication Rounding and resizing.
2. DSP multiplication.
3. Normalisation.
4. Bit reduction and Exponent finder.
5. Post-Multiplication Rounding.

Pre-Multiplication Rounding and resizing

The allocated hardware DSP for multiplication has only 27 bits wide input ports, while the provided input is 29 bits wide, 1 or 2 LSB bits of I and Q parts of samples will be dropped before multiplication to occupy 27 bits depending on the FFT_SIZE. FFT_SIZE of 4096 and 2048 gives an integer part of 13 and 12 respectively, when 1 bit sign and 15 bits fractional parts are taken into account, the total number amounts to 29 and 28 bits respectively. For FFT_SIZE of 4096 two LSB bits are dropped, one bit is truncated while choosing the 28 bits MSB slice then a second LSB bit is dropped in the Rounding sub-block when rounding the sample part to 27 bits. In 2048 FFT_SIZE, one LSB bit is dropped in the Rounding sub-block before DSP input. No LSB bits are dropped for other FFT sizes, this is achieved by choosing the 27 LSB bit slice from the sample since the 2 MSB bits are sign extended. For FFT_SIZEs of 1024 and less, the 27 bits of LSB slice is taken as input of DSP. No rounding is needed in this case, so '0' is appended to LSB and the 28 bits number is passed to the Rounding sub-block to be truncated back to 27 bits without rounding before the DSP input, this is done to avoid an extra if condition and give a uniform operation of Rounding sub-block for all FFT_SIZE cases. As discussed

IQ DSP input	
FFT_SIZE	Format
4096	1.13.13
2048	1.12.14
1024	1.11.15
512	1.11.15
256	1.11.15

Table 4.1: DSP IQ input format for each FFT_SIZE

in the previous chapter, Rounding half up to nearest is achieved by checking the LSB bit of provided input, if it is equal to '1' then 1 is added to the original number. In contrast if LSB bit is equal to '0' then it will be simply truncated and the resulting operation is similar to right shifting the original number by one. The input of Pre-Multiplication Rounding sub-block is always 28 bits and the output is always 27 bits.

Each FFT_SIZE and its corresponding IQ sample format is listed in this table 4.1 Note that for 512 and 256 cases of FFT_SIZE even though the actual integer part length might be up to 10 and 9 bits respectively, the rest of bits will always be sign extended bits.

A psudo code of resizing the IQ signals before Pre-Multiplication Rounding sub-block is depicted in the following listing:

```
1 P_RESIZE:process(CLK)
2 begin
3   if rising_edge(CLK) then
4     if (INT_WIDTH_R0 < "1100") then --reduce 2 MSBs
5       QDATA_R <= signed(DATA_R(29*2 -3 downto 29)) & '0'; -- append '0'
      to be
6       IDATA_R <= signed(DATA_R(29 -3   downto  0)) & '0'; -- truncated
     in rounding
7     elsif (INT_WIDTH_R0 = "1100") then
8       QDATA_R <= signed(DATA_R(29*2 -2 downto 29));
```

```
9       IDATA_R <= signed(DATA_R(29 -2   downto  0));
10    else -- in case (INT_WIDTH_R0 > "1100"), use only MSB bits
11      QDATA_R <= signed(DATA_R(29*2 -1 downto 30));
12      IDATA_R <= signed(DATA_R(29 -1   downto  1));
13    end if;
14  end if;
15 end process P_RESIZE;
```

Listing 4.1: Resize before DSP in DS core

DSP multiplication

To normalise the power of FFT output samples, division over $\sqrt{2}$ is required in two cases, when FFT_SIZE = 2048 and 512. In those two cases, IQ parts of each sample should be multiplied by $\frac{1}{\sqrt{2}}$ factor. For other FFT_SIZE cases, Q and I part of samples are simply multiplied by 1. this multiplication configuration is set independently by Software regardless of FFT_SIZE to give more control to Software.

the DSP operation is built by modifying the original template provided by Intel Quartus to give an optimized block during synthesis.

$\frac{1}{\sqrt{2}} \approx 0.707106781_10 \approx 0.10110101000001001111001100111_2$ converting this to fixed representation format of 1.0.26 to fit into DSP input:

01011010100000100111100110.

To achieve a unified IQ samples format at DSP output for all cases of FFT_SIZE, different multiplication factor format is chosen according to current FFT_SIZE configuration. The 4.2 table: Note that the reciprocal of $\sqrt{2}$ for the smallest three FFT_SIZE cases is also rounded up. The multiplication resulting format for all cases will be 1.15.38, this is because resulting parts width is the addition of both parts from each input, as shown in 4.3 table:

DSP factor			
FFT_SIZE	Format	$\frac{1}{\sqrt{2}}$	1
4096	1.1.25	$00101101010000010011110011_2$	$0100..0_2$
2048	1.2.24	$000101101010000010011110011_2$	$0010..0_2$
1024	1.3.23	$00001011010100000100111010_2$	$0001..0_2$
512	1.3.23	$00001011010100000100111010_2$	$0001..0_2$
256	1.3.23	$00001011010100000100111010_2$	$0001..0_2$

Table 4.2: DSP multiplication factor format for each FFT_SIZE

DSP output format			
FFT_SIZE	IQ format	factor format	output
4096	1.13.13	1.1.25	1.15.38
2048	1.12.14	1.2.24	1.15.38
1024, 512 and 256	1.11.15	1.3.23	1.15.38

Table 4.3: DSP input and output format

Normalisation

Since FFT window size is specified only as Radix-2, dividing the IQ samples at FFT output can be done by right shifting the binary samples bits. The amount of right shifting is derived in the second chapter as $\lfloor Log_2(\sqrt{FFT\_SIZE}) \rfloor$. This results in an integer width of $K =$ INT_WIDTH $- \lfloor Log_2(\sqrt{FFT\_SIZE}) \rfloor$ that the exponent finder operates on to extract an exponent.

Bit reduction and Exponent finder

MSB bits of IQ sample will have sign extended bits after DSP multiplication, these bits are due to the multiplication factor format, as explained previously in Chapter 2. The following example illustrated in figure 4.2 shows one part of a sample (only Q or I) before and after DSP multiplication for different FFT window sizes. The minimum number of bits that can be reduced at the DSP output can be computed from INT_WIDTH value, for FFT_SIZE cases of 256 and 512 i.e. INT_WIDTH of 1001_2 and 1010_2 there should be 2 and 1 sign extended bits respectively. After DSP multiplication sign extended bits are also added from the multiplication factor for all FFT window sizes, these are 2 bits for 4096 FFT_SIZE, 3 bits for 2048 FFT_SIZE and 4 bits for 1024, 512 and 256 window sizes. The maximum number of bits that should be checked for reduction is also dependant on INT_WIDTH value, only integer bits left after applying right shift from normalisation, i.e. K should be checked if they equal the sign bit or differ from sign bit. All those sign extended bits should be reduced to find the exponent.

An example showing reduction of a sample part (I or Q) for FFT window size of 4096 is illustrated in the figure 4.3. The example also shows the compressed Floating-Point format which results at the output of Dynamic Scale, and a Decompression scenario which translates the Floating-Point format back to Fixed-Point format and the quantization error related to it which will be discussed in more details in Performance Chapter 5.

There are two attributes that affects the functionality of exponent finding for an IQ

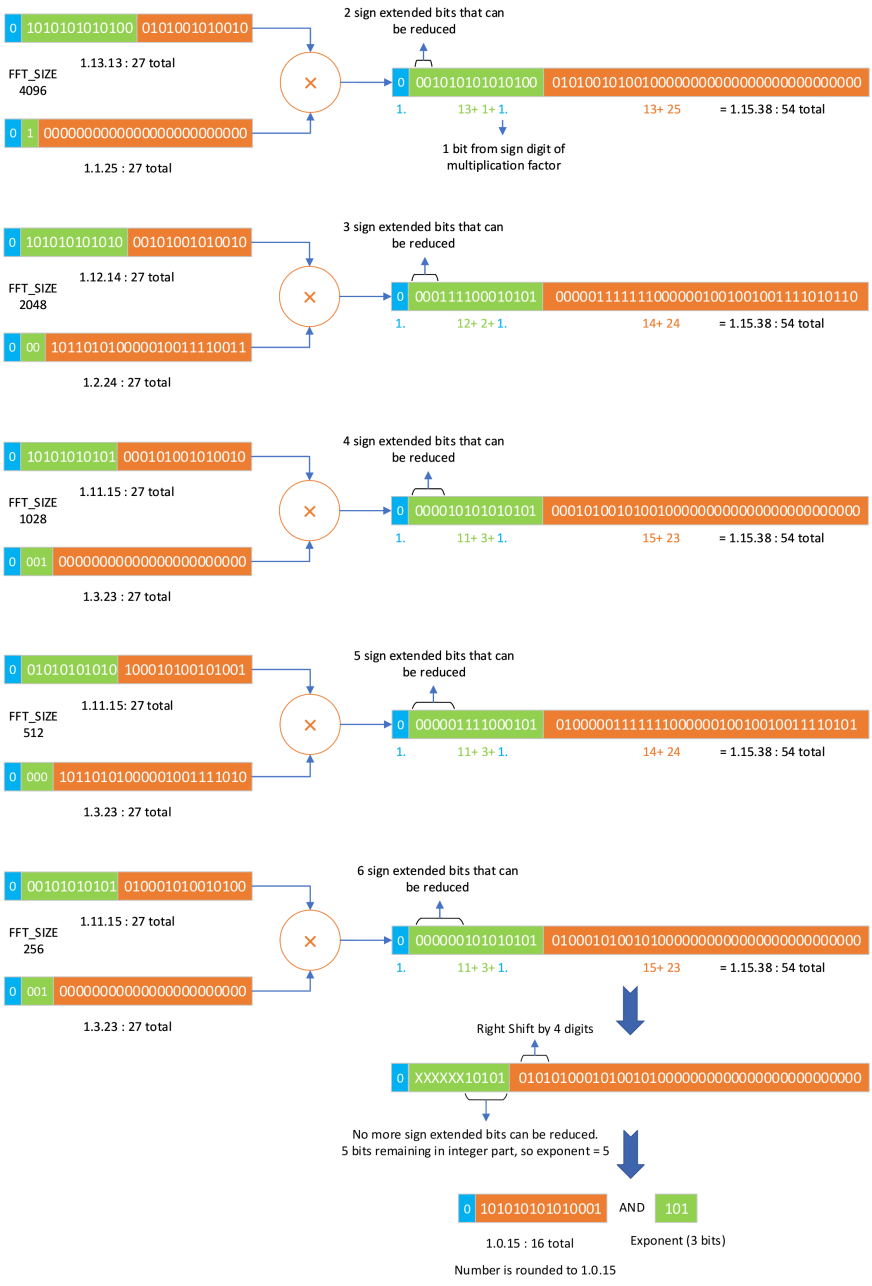


Figure 4.2: DSP input/output bit width for different FFT_SIZE cases

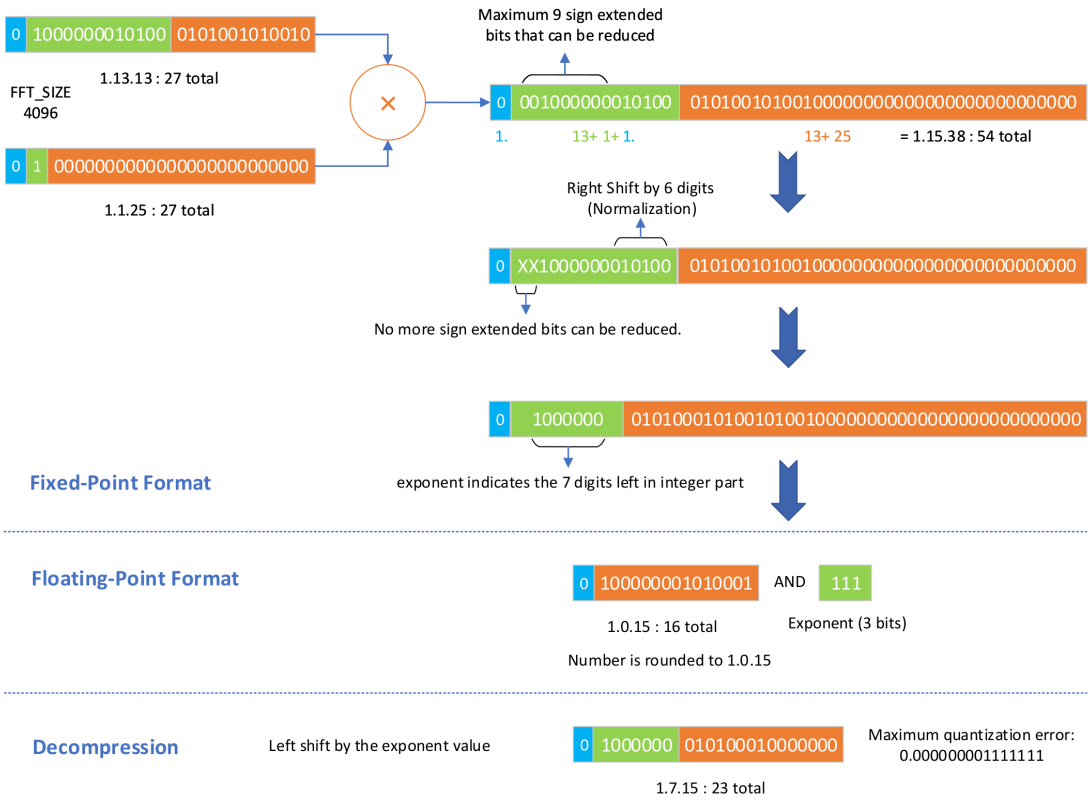


Figure 4.3: Example of Bit reduction for FFT_SIZE case 4096

Compression (FXP -> FLP), and Decompression (FLP -> FXP)

sample, these attributes are INT_WIDTH and right shift Normalisation, both of which can be derived from FFT_SIZE. However, control of these two parameters is left separate to give more Software configuration flexibility.

The following Psudo code snippet gives more detailed functionality of Exponent Finder:

```
P_FINDEXP:process(CLK)
  variable REDUCED_V : integer;
begin
  if (CLK'event and CLK = '1') then
    REDUCED_V := 0;
      for B in 0 to 10 loop -- loop through first 11 MSB bits
        if (QNORM(27*2-2-B) = QNORM(27*2-1) and
            INORM(27*2-2-B) = INORM(27*2-1)) then
            -- if MSB of I and Q are equal to the sign bit
          REDUCED_V := REDUCED_V +1;
        else
          exit -- MAX EXPONENT REACHED
        end if;
      end loop;
    if (INT_WIDTH_I =13 AND REDUCED_V > 9) then -- limit REDUCED_V to 9
        REDUCED_V := 9;
    elsif (INT_WIDTH_I =12 or INT_WIDTH_I =11 and REDUCED_V > 10) then
        REDUCED_V := 10; --limit REDUCED_V to 10
    end if;
    EXPONENT <= 15- REDUCED_V- R_SHIFT_I;
    QNORM_R <= resize(QNORM(27*2 -1 -REDUCED_V downto 0), 27*2);
    INORM_R <= resize(INORM(27*2 -1 -REDUCED_V downto 0), 27*2); --
   choose appropriate chunk and resize it
  end if;
end process P_FINDEXP;
```

Listing 4.2: Finding the sample Exponent in DS core

The line in P_FINDEXP process:

```
EXPONENT_R <= 15- REDUCED_V- R_SHIFT_I;
```

Listing 4.3: Final Exponent

shows that 15- REDUCED_V- R_SHIFT_I is the resulting exponent. REDUCED_V is the number of bits that are sign extended for both I and Q parts of the sample.

Post-Multiplication Rounding

The rounding mechanism chosen in this stage is the same as Pre-Multiplication Rounding stage, i.e. round half-up, this mechanism is chosen due to its resource usage efficiency since it only requires comparing one bit before truncating it. Samples are truncated down to 1.0.16 (leaving one extra LSB bit) then rounded to 1.0.15 by the rounding sub-block. There are two instances of Post-Multiplication Rounding sub-blocks, each corresponding to a different sample part (I or Q). Adding one to the rounded number may cause overflow, this is checked by comparing the high bit of numbers both before and after adding 1. in case the MSB has changed from '0' to '1' then an overflow has occurred, so a SATURATION flag is set to high, and the final results will be rounded by truncation of the specified chunk without adding one.

4.3.2 Common exponent

The block diagram of Common Exponent Scale is shown in figure 4.4. There are N samples at the input each having an exponent of 3 bits, these exponents are first compared to give the maximum exponent in the MAX sub-block. The difference between each samples' exponent and the maximum found exponent is then computed and used to scale the IQ parts of that sample. While down scaling samples with the difference between exponents. Rounding is used in RND sub-block to ensure a low quantization noise in a similar fashion to Dynamic Scale block, i.e. rounding half-up to nearest.

The ports of Common Exponent Scale entity includes:

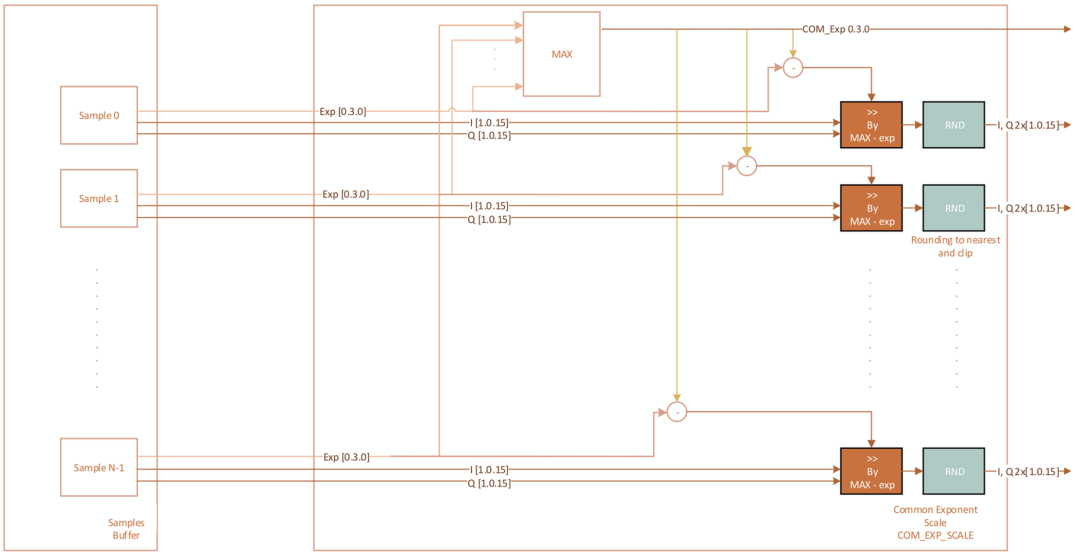


Figure 4.4: Common Exponent Scale block diagram

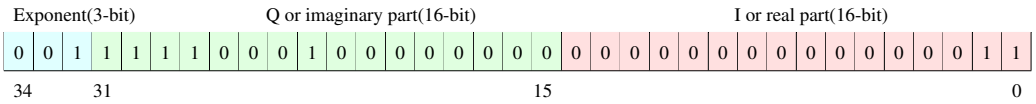


Table 4.4: I, Q and Exponent parts of each sample provided as input for Common Exponent

1. SAMPLE_NUM_G: Generic specifying the number of input samples.
2. DATA_I: Vector of all provided samples.
3. EXPO_O: The maximum exponent out of all provided sample exponents.
4. DATA_O: Vector of all samples after being scaled down and rounded. Only I and Q parts are provided, while the original exponent is discarded.

Each sample of the DATA_I input vector has the format specified in table 4.4.

Common Exponent Scale can be functionally divided into two sub-blocks, these are:

1. MAX

2. Resize.

3. RND.

MAX

The exponent of each sample is extracted and fed into a binary tree of max finder, while IQ parts are sent further for scaling. There is one MAX sub-block instance in Common Exponent Scale for all provided input samples. The input Width of this MAX sub-block corresponds to SAMPLE_NUM_G generic when instantiated.

Resize

To reuse the same Rounding sub-block used in Dynamic Scale Pre and Post DSP multiplication, which always rounds the input width down by 1 bit, IQ samples should be resized accordingly. Only one of the two following cases are possible:

1. Sample should be scaled down by 1 or more bits.

2. Sample does not require any down scaling.

This is found by subtracting the sample's exponent from the maximum found as discussed in the previous chapter. In the first case, the sample is truncated by as much down scaling bits as required minus 1, this LSB will be rounded and cut in RND sub-block. The sample will have a 0 LSB appended in the second case, which will be truncated in RND sub-block without changing the rounding results. The following listing provides a psudo code details of the previous two steps:

```
1 P_COMPEXP:process(clk)
2   variable DIFF_V : integer;-- range 0 to 7;
3   begin
4     if (clk'event and clk = '1') then
5       DIFF_V := 0;
```

```
6       -- compare every sample's exponent with the maximum exponent by
    subtraction.
7       FIND_DIF: for I in 1 to SAMPLE_NUM_G loop
8         DIFF_V := COM_EXP - DATA_R(35*I-1 downto 35*I-3);
9         if DIFF_V > 0 then -- first case.
10          QDATA_R0(I-1)<= resize (QDATA_R(16*I-1 downto 16*I-16+DIFF_V
    -1), 17);
11          IDATA_R0(I-1)<= resize (IDATA_R(16*I-1 downto 16*I-16+DIFF_V
    -1), 17);
12        else -- no right shift required, '0' is appended
13          QDATA_R0(I-1)<= resize (QDATA_R(16*I-1 downto 16*I-16+DIFF_V)
    , 16)&'0';
14          IDATA_R0(I-1)<= resize (IDATA_R(16*I-1 downto 16*I-16+DIFF_V)
    , 16)&'0';
15        end if;
16      end loop;
17    end if;
18  end process;
```

Listing 4.4: Finding the sample Exponent in DS core

RND

There are two Rounding half-up sub-blocks for each IQ sample, one for imaginary part and one for real part. The number of generated pairs of Rounding sub-blocks equals to the number of parallel processed samples.

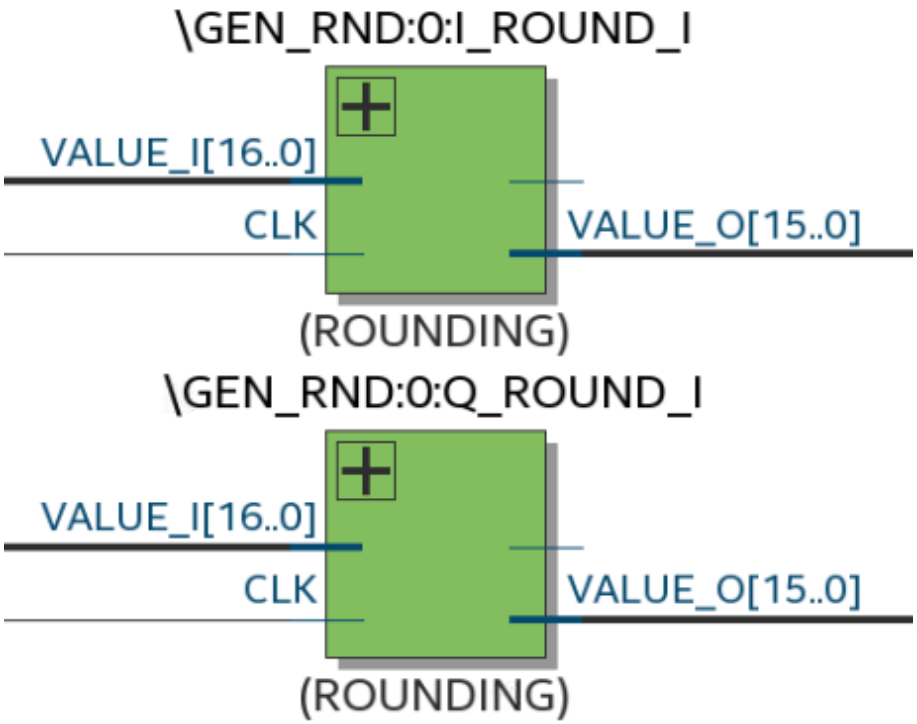


Figure 4.5: Rounding ports block diagram

5 Performance

5.0.1 Synthesis results

Dynamic Scale

The synthesising result for the Dynamic Scale Wrapper Block in terms of ALMs in Quartus is equal to 7,000 from a total of 933,120, meaning that the usage is 0.75% of the total FPGA ALMs. The number of DSPs used in this synthesis is 32 since the number of input samples is 16, where each sample has an imaginary and real parts to be multiplied. The achieved maximum frequency or *Fmax* is 528.82 MHz under operating temperature of 100 Degrees Celsius and 850 mV. A single Dynamic Scale block has utilisation of 354 ALMs and 2 DSP blocks. Total Thermal Power Dissipation in Dynamic Scale Wrapper (total) equals to 21821.99 mW. While the Core Dynamic Thermal Power Dissipation is 465.05 mW.

Common Exponent Scale

When synthesising the Common Exponent Scale Block for 16 input samples the achieved ALM results is 3,116 out of 933,120 which is around 0.33% usage of the total FPGA ALMs. The maximum frequency *Fmax* achieved under stress conditions of 100 Degrees Celsius for 850 milli-Volts is 627.35 MHz. Total Thermal Power Dissipation for Common Exponent Scale equals to 21643.16 mW. While the Core Dynamic Thermal Power Dissipation is 302.28 mW.

More details can be found in the Appendix A.

IP block	ALMs	Area	DSP	Fmax	Total Power
Dynamic Scale	7000	0.75%	32	528.82 MHz	21821.99 mW
Common Exponent	3116	0.33%	0	627.35 MHz	21643.16 mW

Table 5.1: Synthesis results

5.0.2 Matlab

Matlab offers 5G toolbox for testing throughput of Public Shared Channel (PUSCH) in up-link path. [17] The Dynamic scale C model is integrated to the path using a Matlab executable (MEX) wrapper. The results of FFT function is tunnelled into Dynamic scale to test the quantization and functionality. Because the C model should give a bit accurate representation, IQ parts of samples at the input are declared as *int* type instead of *float*, this allows a maximum of 32 bits for each I and Q separately, while the Hardware input ports have a maximum of 29 bits for each. IQ samples at FFT output theoretically should not exceed 29 bits for FFT window size of 4096 or less. Matlab FFT function gives complex float numbers as output, to cast those float samples into integers with 15 bits for the fractional part as in the previously mentioned format in 2.1, IQ parts of samples are multiplied with 2^{15} then fed into the mex wrapper. The mex wrapper also takes the FFT window size as a parameter in order to normalize samples correctly. At the output of Dynamic scale, Samples are divided by $2^{15-\lfloor Log_2(\sqrt{FFT\_SIZE})\rfloor}$ to cast them back to floating numbers with the corresponding power.

Throughput

Matlab 5G toolbox testing throughput performance has an FFT window size of 1024 samples. The simulation is ran for 5 frames of 10 ms for SNR range [-3 8] with a step of 1 and HARQ re-transmission disabled. Due to the limited control parameters that can be defined in the toolbox provided by Matlab, signals replicating a multiple UEs scenario

cannot be tested, thus the FFT output is limited to a maximum of 2 digits for the integer part of IQ samples. i.e. the range of samples at FFT output is $]-3.000, 3.000[$ for both imaginary and real parts. By default these numbers are not large enough to produce an exponent to test the quantization effect of dynamic scale, instead, to test the effect of dynamic scale block on samples, less bits of sample's fractional part are used. for example a 256 QAM and 1024 FFT size has the following complex float sample at the output $2.321 + i1.922$ which is equal to $10.010100100010110100001110010 1...{}_2 + i1.11101100000010000011000100100...{}_2$ in binary. Full precision of Fixed-point has 15 bits given for the fractional part thus the sample becomes $10.0101001000101101_2 + i1.1110110000001000_2$ or $2.320 + i1.921$ in decimal, introducing a quantization error of 0.001. Reducing the number of fractional part digits using Dynamic scale to measure quantization for 1024 FFT size and 256 QAM from 15 bits to 1 bit gives the results in figure 5.1. The figure shows that when using only 1 bit for the fractional part the throughput does not reach reach 100 since the quantization error is very large. Using 2 bits achieves a throughput of 100 at higher Signal-to-Noise Ratio (SNR) than using more bits. When using 3 bits or 4 bits as fractional part the same throughput is achieved, which falls slightly behind the full precision floating point in SNR ranging from 4 to 7. When using 5 bits or more to represent the sample's fractional part, the same throughput is achieved as using full precision floating point.

Quantization

When reducing the amount of bits used in the fractional part more quantization is introduced, this is due to loss in precision when representing fixed-point numbers using floating-point numbers. Matlab offers a quantizer function that defines the number of bits for both integer and fractional parts using a fixed-point representation on input to introduce quantization noise. *quantizer([11 5], 'fixed', 'nearest');*. When 5 bits for the fractional part are specified for both Matlab Quantizer and Dynamic Scale input the result-

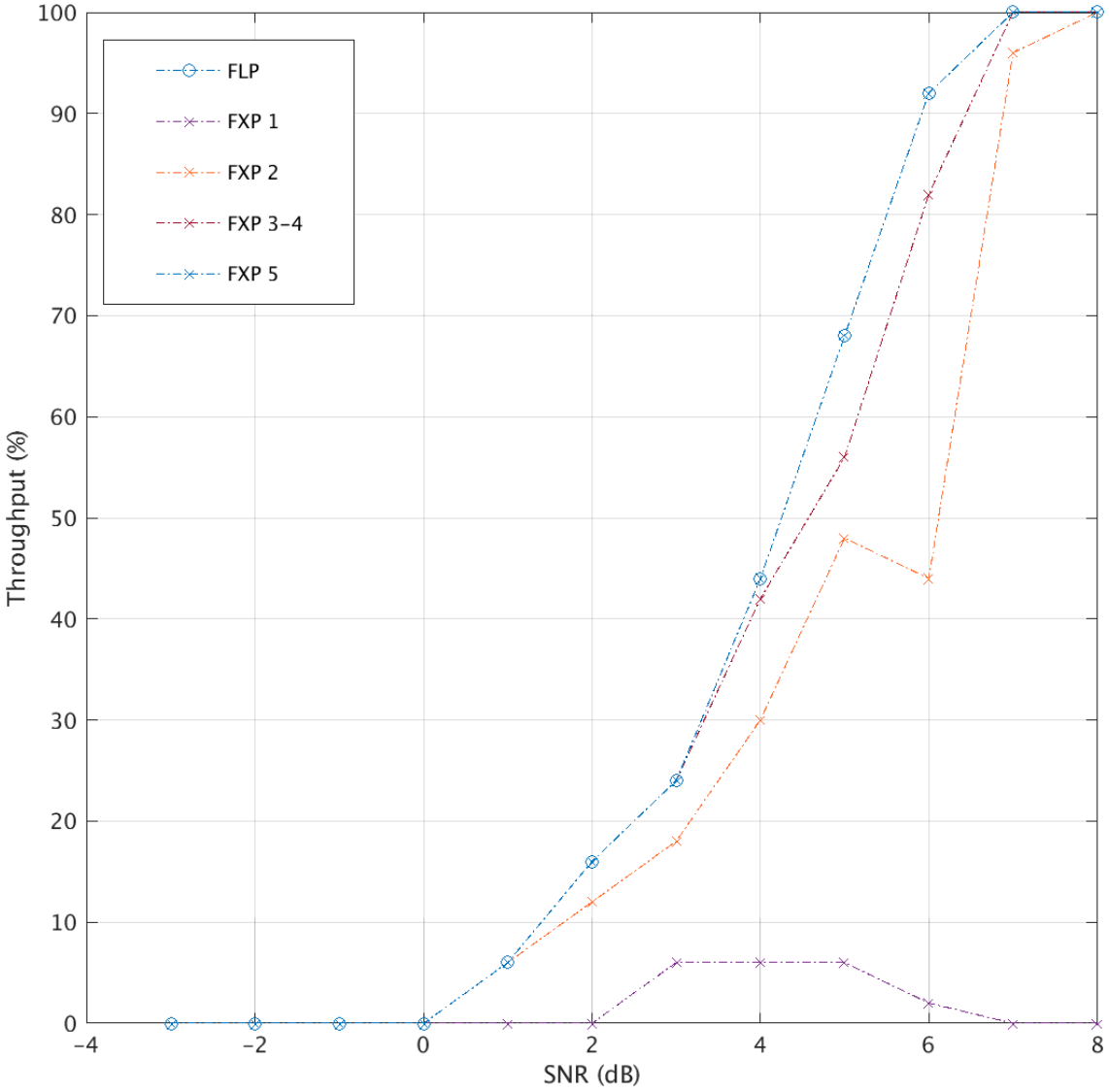


Figure 5.1: Dynamic scale block diagram

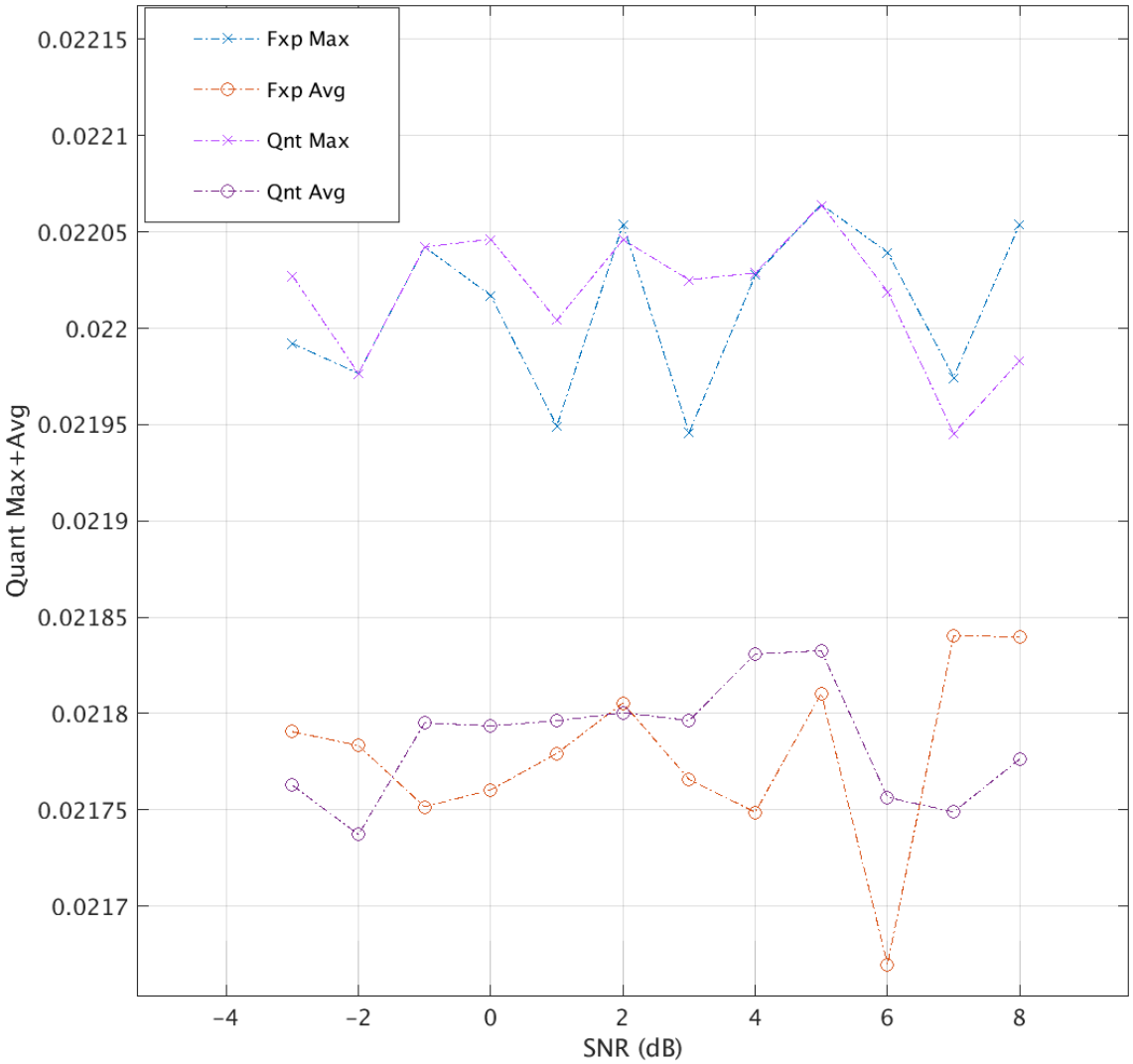


Figure 5.2: Dynamic scale block diagram

ing quantization error can be viewed in figure 5.2. Five bits are chosen for the fractional part because it provides the same throughput as the full precision floating-point during the simulation. The Dynamic scale fixed-point maximum and average quantization for all 5 frames to each SNR is collected and compared against Matlab quantize function. The results in figure 5.2 shows that using Dynamic scale introduces less quantization error on average for SNR between -1 to 6. While achieving also less maximum quantization error than Matlab for SNR range from 0 to 4 dB.

6 Discussion

6.1 Conclusion

Careful planning and methods of approaching the solution of any problem can mitigate bad implementation decisions. This thesis work started by defining the basic mathematical concepts in $5G$ and LTE to give a good perspective on where the project can be deployed, then the mathematical building blocks of needs implementing were explained in details, and the Hardware constraints imposed by the platform architecture were also defined, and finally the implementation itself was discussed. During the implementation phase, many changes and modifications were made to the design to give better overall performance, this required early performance testing and verification, so test-benches and C reference models integrated into Matlab were developed in parallel with the RTL design. The downside of this is that any design changes repelled through all three stages of C reference Model, Matlab, and of course the RTL design itself.

6.2 Further work

A more thorough and complex testing scenarios that covers wider signals spectrum can be implemented to test the effects of higher exponents on the overall SNR of the system, however this partially done on proprietary Nokia tools and the results were not authorised to be shared in the scope of this thesis.

References

[1] M. A. A. M. J. S. Shahabuddin, "Massive mimo detection techniques: A survey", 2019. DOI: 10.1109/COMST.2019.2935810.

[2] D. A. Basnayaka and H. Haas, "Spatial modulation for massive mimo", pp. 1945–1950, 2015.

[3] T. L. Marzetta, "Noncooperative cellular wireless with unlimited numbers of base station antennas", vol. 9, pp. 3590–3600, 2010.

[4] M. D. E. Björnson E. Larsson, "Massive mimo for maximal spectral efficiency: How many users and pilots should be allocated?", vol. 15, pp. 1293–1308, 2016.

[5] e. a. H. Chen J. Chen, "Spectral-energy efficiency trade-off in relay-aided massive mimo cellular networks with pilot contamination", vol. 4, pp. 5234–5242, 2016.

[6] D. P. e. a. F. Rusek, "Scaling up mimo: Opportunities and challenges with very large arrays", vol. 30, pp. 40–60, 2013.

[7] e. a. T. Do E. Björnson, "Jamming-resistant receivers for the massive mimo uplink", vol. PP, pp. 210–223, 2017.

[8] H. Alshamary, "Coherent and non-coherent data detection algorithms in massive mimo", 2017.

[9] e. a. S. Jin K. Wong, "On massive mimo zero-forcing transceiver using time-shifted pilots", vol. 65, pp. 59–74, 2016.

[10] "145-1993 - ieee standard definitions of terms for antennas", 2014.

[11] M. Gast, “802.11 wireless networks: The definitive guide”, p. 284, 2005.

[12] IEEE, “754-1985 - ieee standard for binary floating-point arithmetic”, 1990.

[13] G. B. A. H. J. Weber, “Mathematical methods for physicists”, 2001.

[14] D. Elam and C. Iovescu, “A block floating point implementation for an n-point fft on the tms320c55x dsp”, 2003.

[15] Intel, “Intel stratix 10 device datasheet”, 2019.

[16] ——, “Quartus prime pro edition help version 19.4”, Adaptive Logic Module (ALM) Definition, 2019.

[17] MATLAB, *NR PUSCH Throughput, version 9.7.0 (R2019b)*. Natick, Massachusetts: The MathWorks Inc., 2019.

Appendix A Resource Usage

A.1 Dynamic Scale

Resource	Usage	%
Logic utilization (ALMs needed / total ALMs on device)	6,579 / 933,120	< 1 %
ALMs needed [=A-B+C]	6,579	
[A] ALMs used in final placement [=a+b+c+d]	5,799 / 933,120	< 1 %
[a] ALMs used for LUT logic and register circuitry	2,142	
[b] ALMs used for LUT logic	1,296	
[c] ALMs used for register circuitry	2,341	
[d] ALMs used for memory (up to half of total ALMs)	20	
[B] Estimate of ALMs recoverable by dense packing	1 / 933,120	< 1 %
[C] Estimate of ALMs unavailable [=a+b+c+d]	781 / 933,120	< 1 %
[a] Due to location constrained logic	0	
[b] Due to LAB-wide signal conflicts	0	
[c] Due to LAB input limits	0	
[d] Due to virtual I/Os	781	
Difficulty packing design	N/A	
Total LABs: partially or completely used	582 / 93,312	< 1 %
-- Logic LABs	580	
-- Memory LABs (up to half of total LABs)	2	
Combinational ALUT usage for logic	6,874	
-- 8 input functions	0	
-- 7 input functions	0	
-- 6 input functions	3,392	
-- 5 input functions	1,072	
-- 4 input functions	256	
-- <=3 input functions	2,154	
Combinational ALUT usage for route-throughs	0	
Memory ALUT usage	39	
-- 64-address deep	0	
-- 32-address deep	39	
Dedicated logic registers	9,029	
-- By type:		
-- LAB logic registers:		
-- Primary logic registers	8,990 / 1,866,24	< 1 %

Figure A.1: Dynamic Scale Wrapper Resource Usage

Resource	Usage	%
-- Secondary logic registers	0 / 1,866,240	0 %
-- Hyper-Registers:	39	
Register control circuitry for power estimation	0	
ALMs adjustment for power estimation	-1	
I/O pins	0 / 912	0 %
-- Clock pins	0 / 72	0 %
-- Dedicated input pins	3 / 102	3 %
Hard processor system peripheral utilization		
-- Clock resets	0 / 1 (0 %)	
-- Cross trigger	0 / 1 (0 %)	
-- S2F AXI	0 / 1 (0 %)	
-- F2S AXI	0 / 1 (0 %)	
-- AXI Lightweight	0 / 1 (0 %)	
-- SDRAM	0 / 1 (0 %)	
-- Interrupts	0 / 1 (0 %)	
-- JTAG	0 / 1 (0 %)	
-- Loan I/O	0 / 1 (0 %)	
-- MPU event standby	0 / 1 (0 %)	
-- MPU general purpose	0 / 1 (0 %)	
-- STM event	0 / 1 (0 %)	
-- TPIU trace	0 / 1 (0 %)	
-- DMA	0 / 1 (0 %)	
-- EMAC	0 / 3 (0 %)	
-- I2C	0 / 5 (0 %)	
-- NAND Flash	0 / 1 (0 %)	
-- SDMMC	0 / 1 (0 %)	
-- SPI Master	0 / 2 (0 %)	
-- SPI Slave	0 / 2 (0 %)	
-- UART	0 / 2 (0 %)	
M20K blocks	0 / 11,721	0 %
Total MLAB memory bits	312	
Total block memory bits	0 / 240,046,080	0 %

Figure A.2: Dynamic Scale Wrapper Resource Usage

Resource	Usage	%
Total block memory implementation bits	0 / 240,046,080	0 %
DSP Blocks Needed [=A+B-C]	32 / 5,760	< 1 %
[A] Total Fixed Point DSP Blocks	32	
[B] Total Floating Point DSP Blocks	0	
[C] Estimate of DSP Blocks recoverable by dense merging	0	
IOPLLs	0 / 14	0 %
Global signals	1	
Impedance control blocks	0 / 24	0 %
Maximum fan-out	9113	
Highest non-global fan-out	944	
Total fan-out	50351	
Average fan-out	2.87	

Figure A.3: Dynamic Scale Wrapper Resource Usage

A.2 Common Exponent Scale

Resource	Usage	%
Logic utilization (ALMs needed / total ALMs on device)	15,816 / 933,120	2 %
ALMs needed [=A-B+C]	15,816	
[A] ALMs used in final placement [=a+b+c+d]	14,887 / 933,120	2 %
[a] ALMs used for LUT logic and register circuitry	4,422	
[b] ALMs used for LUT logic	4,609	
[c] ALMs used for register circuitry	5,856	
[d] ALMs used for memory (up to half of total ALMs)	0	
[B] Estimate of ALMs recoverable by dense packing	1,256 / 933,120	< 1 %
[C] Estimate of ALMs unavailable [=a+b+c+d]	2,185 / 933,120	< 1 %
[a] Due to location constrained logic	0	
[b] Due to LAB-wide signal conflicts	0	
[c] Due to LAB input limits	7	
[d] Due to virtual I/Os	2,178	
Difficulty packing design	High	
Total LABs: partially or completely used	1,728 / 93,312	2 %
-- Logic LABs	1,728	
-- Memory LABs (up to half of total LABs)	0	
Combinational ALUT usage for logic	10,659	
-- 8 input functions	126	
-- 7 input functions	0	
-- 6 input functions	5,945	
-- 5 input functions	2,154	
-- 4 input functions	131	
-- <=3 input functions	2,303	
Combinational ALUT usage for route-throughs	11,304	
Dedicated logic registers	32,004	
-- By type:		
-- LAB logic registers:		
-- Primary logic registers	20,555 / 1,866,240	1 %
-- Secondary logic registers	9,258 / 1,866,240	< 1 %
-- Hyper-Registers:	2,191	
Register control circuitry for power estimation	349	

1

Figure A.4: Common Exponent Scale Resource Usage

Resource	Usage	%
ALMs adjustment for power estimation	722	
I/O pins	0 / 912	0 %
-- Clock pins	0 / 72	0 %
-- Dedicated input pins	3 / 102	3 %
Hard processor system peripheral utilization		
-- Clock resets	0 / 1 (0 %)	
-- Cross trigger	0 / 1 (0 %)	
-- S2F AXI	0 / 1 (0 %)	
-- F2S AXI	0 / 1 (0 %)	
-- AXI Lightweight	0 / 1 (0 %)	
-- SDRAM	0 / 1 (0 %)	
-- Interrupts	0 / 1 (0 %)	
-- JTAG	0 / 1 (0 %)	
-- Loan I/O	0 / 1 (0 %)	
-- MPU event standby	0 / 1 (0 %)	
-- MPU general purpose	0 / 1 (0 %)	
-- STM event	0 / 1 (0 %)	
-- TPIU trace	0 / 1 (0 %)	
-- DMA	0 / 1 (0 %)	
-- EMAC	0 / 3 (0 %)	
-- I2C	0 / 5 (0 %)	
-- NAND Flash	0 / 1 (0 %)	
-- SDMMC	0 / 1 (0 %)	
-- SPI Master	0 / 2 (0 %)	
-- SPI Slave	0 / 2 (0 %)	
-- UART	0 / 2 (0 %)	
M20K blocks	0 / 11,721	0 %
Total MLAB memory bits	0	
Total block memory bits	0 / 240,046,080	0 %
Total block memory implementation bits	0 / 240,046,080	0 %
DSP Blocks Needed [=A+B-C]	0 / 5,760	0 %
[A] Total Fixed Point DSP Blocks	0	

Figure A.5: Common Exponent Scale Resource Usage

Resource	Usage	%
[B] Total Floating Point DSP Blocks	0	
[C] Estimate of DSP Blocks recoverable by dense merging	0	
IOPLLs	0 / 14	0 %
Global signals	1	
Impedance control blocks	0 / 24	0 %
Average interconnect usage (total/H/V)	0.2% / 0.2% / 0.2%	
Peak interconnect usage (total/H/V)	8.8% / 8.5% / 10.6%	
Maximum fan-out	32353	
Highest non-global fan-out	50	
Total fan-out	119627	
Average fan-out	2.52	

Figure A.6: Common Exponent Scale Resource Usage