Fixed-Point FPGA Implementation of the FFT Accumulation Method for Real-time Cyclostationary Analysis

CAROL JINGYI LI, The University of Sydney, Faculty of Engineering, School of Electrical and Information Engineering, Australia
XIANGWEI LI, Nanyang Technological University, School of Computer Science and Engineering, Singapore
BINGLEI LOU, The University of Sydney, Faculty of Engineering, School of Electrical and Information Engineering, Australia
CRAIG T. JIN, The University of Sydney, Faculty of Engineering, School of Electrical and Information Engineering, Australia
DAVID BOLAND, The University of Sydney, Faculty of Engineering, School of Electrical and Information Engineering, Australia
PHILIP H.W. LEONG, The University of Sydney, The University of Sydney Nano Institute, Faculty of Engineering, School of Electrical and Information Engineering, Australia

The spectral correlation density (SCD) is an important tool in cyclostationary signal detection and classification. Even using efficient techniques based on the fast Fourier transform (FFT), real-time implementations are challenging because of the high computational complexity. A key dimension for computational optimization lies in minimizing the wordlength employed. In this paper, we analyze the relationship between wordlength and signal-to-quantization noise in fixed-point implementations of the SCD function. A canonical SCD estimation algorithm, the FFT accumulation method (FAM) using fixed-point arithmetic is studied. We derive closed-form expressions for SQNR and compare them at wordlengths ranging from 14 to 26 bits. The differences between the calculated SQNR and bit-exact simulations are less than 1 dB. Furthermore, an HLS-based FPGA design is implemented on a Xilinx Zynq UltraScale+ XCZU28DR-2FFVG1517E RFSoC. Using less than 25% of the logic fabric on the device, it consumes 7.7 W total on-chip power and has a power efficiency of 12.4 GOPS/W, which is an order of magnitude improvement over an Nvidia Tesla K40 graphics processing unit (GPU) implementation. In terms of throughput, it achieves 50 MS/sec, which is a speedup of 1.6 over a recent optimized FPGA implementation.

CCS Concepts: • **Hardware → Hardware accelerators**; **Signal processing systems**; • **Mathematics of computing → Arbitrary-precision arithmetic**.

Additional Key Words and Phrases: SCD, FAM, quantization error, HLS, FPGAs.

Authors' addresses: Carol Jingyi Li, jingyi.li@sydney.edu.au, The University of Sydney, Faculty of Engineering, School of Electrical and Information Engineering, Sydney, New South Wales, Australia, Sydney, Australia, 2006; Xiangwei Li, xli045@e.ntu.edu.sg, Nanyang Technological University, School of Computer Science and Engineering, 50 Nanyang Avenue, Singapore, 639798; Binglei Lou, The University of Sydney, Faculty of Engineering, School of Electrical and Information Engineering, Sydney, New South Wales, Australia, Sydney, Australia, binglei.lou@sydney.edu.au; Craig T. Jin, The University of Sydney, Faculty of Engineering, School of Electrical and Information Engineering, Sydney, New South Wales, Australia., Sydney, Australia, craig.jin@sydney.edu.au; David Boland, The University of Sydney, Faculty of Engineering, School of Electrical and Information Engineering, Sydney, New South Wales, Australia, Sydney, Australia, david.boland@sydney.edu.au; Philip H.W. Leong, The University of Sydney, The University of Sydney Nano Institute, Faculty of Engineering, School of Electrical and Information Engineering, Sydney, New South Wales, Australia, Sydney, Australia, philip.leong@sydney.edu.au.

1936-7406/2022/9-ART111 $15.00
https://doi.org/XXXXXXX.XXXXXXX

ACM Reference Format:
Carol Jingyi Li, Xiangwei Li, Binglei Lou, Craig T. Jin, David Boland, and Philip H.W. Leong. 2022. Fixed-Point FPGA Implementation of the FFT Accumulation Method for Real-time Cyclostationary Analysis. *ACM Trans. Reconfig. Technol. Syst.* 37, 4, Article 111 (September 2022), 30 pages. https://doi.org/XXXXXXX.XXXXXXX

GLOSSARY OF SYMBOLS

T_s	sampling period	N	number of samples in a window
Δf	frequency resolution	$\Delta\alpha$	cyclic frequency resolution
G_*	gain of sections ($*$ refers to section)	$P_{i.s}$	power of input signal
N_c	$2^{\log_2(T_s/\Delta f)}$	N_f	$2^{\log_2(T_s/\Delta\alpha/L)}$
L	$N_c/4$	P	$\lfloor N/L \rfloor$
m_1	$\log_2(N_c)$	m_2	$\log_2(N_f)$
A_1	$2 - \frac{m_1+1.5}{N_c}$	A_2	$2 - \frac{m_2+1.5}{N_f}$
B_1	$\frac{2^{m_1}}{6} - 1$	B_2	$\frac{2^{m_2}}{6} - 1$
q_1	normalization coefficient for the first FFT		
q_2	normalization coefficient for the conjugate multiplication		
q_3	normalization coefficient for the second FFT		

1 INTRODUCTION

A time series is said to be *cyclostationary* if its probability distribution varies periodically with time. Cyclostationary time series analyses are suitable for a wide range of periodic phenomena in signal processing, including characterization of modulation types; noise analysis of periodic time-variant linear systems; synchronization problems; parameter and waveform estimation; channel identification and equalization; signal detection and classification; AR and ARMA modelling and prediction; and source separation [8]. A signal exhibits cyclostationarity if and only if the signal is correlated with certain frequency-shifted versions of itself [6]. Cyclostationary analysis often involves computing the spectral correlation density (SCD), also called the cyclic spectral density or spectral correlation function. This function describes the cross-spectral density of all pairs of frequency-shifted versions of a time-series.

Although the SCD method reveals rich information about cyclostationary processes even under low signal-to-noise ratio conditions, its high computational complexity makes it difficult to apply in real-time applications. Thus there has been interest in developing high-performance implementations of the FAM method to detect and classify cyclostationary signals on processors (CPUs) [19], graphics processing units (GPUs) [14] and field programmable gate arrays (FPGAs) [2, 13].

To maximise performance, fixed-point implementations of signal processing techniques should be considered as they are more computationally efficient than floating point implementations and can lead to improved hardware efficiency, at the cost of a quantization error. In this work, we analyse the performance of the FAM method in terms of signal to quantization noise ratio (SQNR), introduce two methods of fixed-point design, namely Fixed Precision (FAM_M1) and Mixed Precision (FAM_M2), and compare their performance.

Furthermore, we present an open-source[1], scalable high-speed FPGA-based SCD accelerator, utilizing on-chip high speed arithmetic primitives present in the digital signal processing (DSP) and memory blocks to verify the calculation result. The design is synthesized from a C description using high-level synthesis (HLS) tools [15], allowing our calculations to be verified and performance metrics such as speed and performance to be determined. Previous work in references [5, 14, 17] use floating-point calculations and are unable to achieve the performance of our proposed design.

[1]https://github.com/Jingyi-li/FAM_Synthesis.git

A summary of our contributions are:

- The first analytical SQNR model for fixed-point implementations of the FFT accumulation method (FAM) technique for estimating SCD, enabling aggressive tradeoffs between precision and area.
- A quantitative comparison of two wordlength assignment strategies, FAM_M1 which employs a fixed wordlength throughout the data path, and FAM_M2 with mixed precision.
- A parallel architecture for computing the SCD using fixed-point arithmetic with the FAM_M1 and FAM_M2 wordlength assignments. The architecture is highly parallel and allows mixed precisions to be used throughout.
- An HLS implementation of the architecture which, to the best of our knowledge, achieves the highest reported throughput and power performance for the FAM technique. It employs our SQNR model to minimise resource requirements through careful precision optimisation and sparse matrix output to minimise accelerator-to-host bandwidth.

The remainder of this paper is organised as follows. In Section 2 we provide background on SCD analysis, the theory statistical quantization analysis and Vivado HLS [11]. In Section 3, our error analysis is given. The implementation of our HLS implementation of a FAM method is presented in Section 4. In Section 5, a comparison of our theory with simulations and the performance of our FPGA implementation is detailed. Finally, we draw conclusions in Section 6.

2 BACKGROUND

2.1 Spectral Correlation Density Estimation

The SCD function can be implemented either via time or frequency smoothing. In this paper we focus on time smoothing, since it has been shown to be more computationally efficient in general [17]. There are two common time-smoothing methods, the FFT accumulation method (FAM) [17, 19] and the strip spectral correlation algorithm (SSCA) [17, 20].

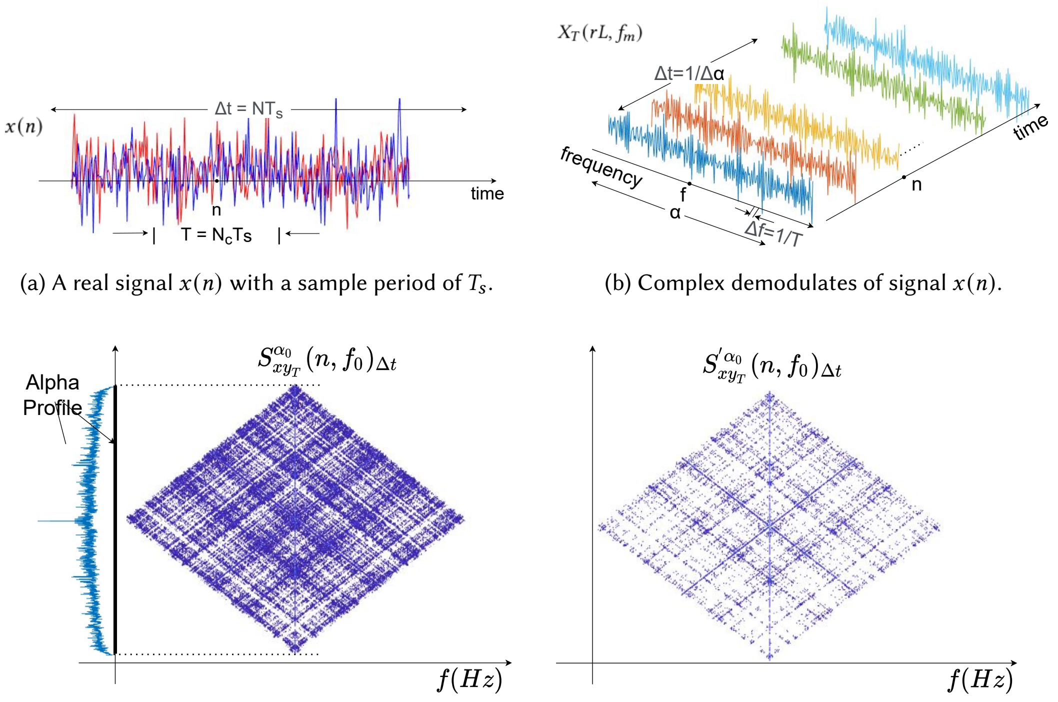
Due to its parallel FFT-based computation and regular data access patterns, the FAM technique is a commonly used estimator of the SCD, particularly for a small number of cycle frequencies. The SSCA method is more efficient when all cycle frequencies are of interest, and Antoni's fast spectral correlation estimator [1] has advantages over FAM and SSCA when the maximum considered cycle frequency is a small fraction of the sampling rate [21]. The techniques proposed in this paper could be extended to these and others in a straightforward manner.

The description of the SCD function below follows that of Roberts et. al. [17] and Brown et. al [5]. The discrete-time *complex demodulate* of a continuous time, complex-valued signal $x(t)$ at frequency f is

$$X_T(n,f) = \sum_{r=-N/2}^{N/2} a(r)x(n-r)e^{-i2\pi f(n-r)T_s} \tag{1}$$

where $a(r)$ is a length $T = NT_s$ second windowing function, T_s is the sampling period and N is the number of samples. Complex demodulates are low pass sequences with bandwidths $\Delta f \approx 1/T$. For inputs $x(n)$ and $y(n)$ of length N samples, we correlate demodulates $X_T(n, f_1)$ and $Y_T(n, f_2)$ separated by α_0 ($f_1 = f_0 + \alpha_0/2$, $f_2 = f_0 - \alpha_0/2$) over the time window $\Delta t = NT_s$ using a complex multiplier followed by a low pass filter (LPF) with bandwidth approximately $1/\Delta t$. Thus the SCD function is given by

$$S_{xy_T}^{\alpha_0}(n, f_0)_{\Delta t} = \sum_r X_T(r, f_1)Y_T^*(r, f_2)g(n-r) \tag{2}$$



(a) A real signal $x(n)$ with a sample period of T_s.

(b) Complex demodulates of signal $x(n)$.

(c) The SCD function of signal $x(n)$.

(d) Sparse SCD with only 0.457% non-zero values.

Fig. 1. The SCD function and sparse SCD of OOK signal from DeepSig [10] at SNR = -8 dB.

where the * operator is a complex conjugate and $g(n)$ is a length $\Delta t = NT_s$ windowing function. For the special case of auto-correlation studied in this paper, $y(n)$ is a time-delayed value of $x(n)$, i.e., $y(n) = x(n + d)$ where d is the delay.

2.2 Example of Spectral Correlation Desity, Alpha Profile and Sparse SCD

Fig. 1 shows an on-off keying (OOK) modulated in-phase and quadrature (I/Q) signal $x(n)$ with *complex demodulate* $X_T(rL, f_m)$, an estimated SCD function $S^{\alpha_0}_{xy_T}(n, f_0)_{\Delta t}$, the alpha profile and the sparse SCD $S'^{\alpha_0}_{xy_T}(n, f_0)_{\Delta t}$. The alpha profile indicates maximal SCD values along the alpha frequency axis and is defined by:

$$\text{alpha profile} = \max_{\alpha_* \in \alpha_0}(S^{\alpha_*}_{xy_T}(n, f_0)_{\Delta t}). \tag{3}$$

The sparse SCD matrix is formed from the full SCD matrix by setting values smaller than a threshold value ($Tred$) to zero. It captures the critical information and greatly reduces storage requirements compared to the full SCD matrix. The sparse SCD is defined as:

$$S'^{\alpha_0}_{xy_T}(n, f_0)_{\Delta t} = \begin{cases} S^{\alpha_0}_{xy_T}(n, f_0)_{\Delta t} & \text{if } (S^{\alpha_0}_{xy_T}(n, f_0)_{\Delta t} \geq Tred) \\ 0 & \text{otherwise} \end{cases} . \tag{4}$$

To the best of our knowledge, this work is the first to use a sparse SCD output format.

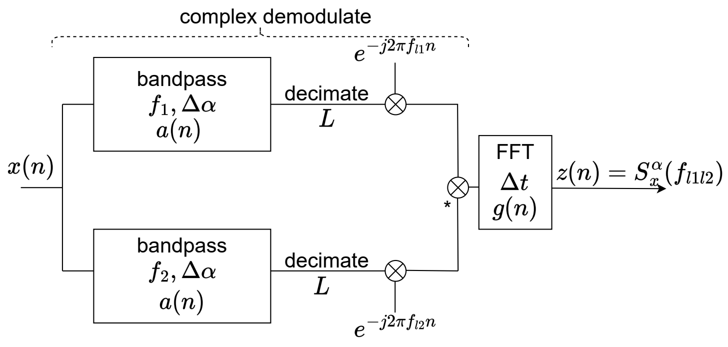


Fig. 2. The FFT accumulation method (FAM)

2.3 FAM Technique

The direct application of Eq. (2) is computationally inefficient. Decimation and the fast Fourier transform (FFT) can be used to reduce the computational complexity [17]. Fig. 2 illustrates the signal flow for the FAM method, where the first task for both methods is to compute the complex demodulates, X_T and Y_T (in Eq. (2)). We summarize the computations in this subsection, and refer readers to references [5, 7] for a detailed derivation.

2.3.1 Complex Demodulate. For Eq. (1), the input sequence is from N to $P = N/L$ via decimation using an L sample stride for the channelizer, with $L = N_c/4$ [5]. The Eq. (1) can be rewritten as

$$X_T(pL, f) = \sum_{r=-N_c/2}^{N_c/2} a(r)x(pL - r)e^{-i2\pi f(pL-r)T_s} \tag{5}$$

where $p = \{0, 1..., P - 1\}$. Substituting $d = N_c/2 - 1, r = d - k, f_m = mf_s/N_c$ and $-N_c/2 < m < N_c/2$ [7], Eq. (5) becomes

$$\begin{aligned} X_T(pL, f_m) &= \sum_{k=0}^{N_c-1} a(d-k)x(pL-d+k)e^{-i2\pi m(pL-d+k)/N_c} \\ &= [\sum_{k=0}^{N_c-1} a(d-k)x(pL-d+k)e^{-i2\pi mk/N_c}]e^{-i2\pi m(pL-d)/N_c} \\ &= \underbrace{[\sum_{k=0}^{N_c-1} \underbrace{a(d-k)x(pL-d+k)}_{x(n) \text{ windowed by } a(n)} e^{-i2\pi mk/N_c}]}_{N_c \text{ point-FFT}} \underbrace{e^{-i2\pi mpL/N_c}}_{\text{Down Conversion}} . \end{aligned} \tag{6}$$

Thus, according to Eq. (6), the input is windowed via $a(n)$, then passed through a N_c-Point FFT. A phase shift is introduced to compensate for the down conversion from N to N_c samples.

2.3.2 FAM method. Taking Eq. (2), and substituting $X_T = Y_T$ to compute at the frequency $f_{kl} = (f_k + f_l)/2$, Eq. (2) becomes

$$S_{xy_T}^{\alpha_0}(n, f_{kl})_{\Delta t} = \sum_r X_T(r, f_k)X_T^*(r, f_l)g(n-r). \tag{7}$$

Applying this for P segments (Eq. (6)), the SCD function becomes

$$S^{\alpha_0}_{xy_T}(pL, f_{kl})_{\Delta t} = \sum_r X_T(rL, f_k) X^*_T(rL, f_l) g_d(p - r) \tag{8}$$

where $p = \{0, 1, ..., P-1\}$ and $g_d(r) = g(rL)$. Now the cycle frequency parameter has been redefined to $\alpha_0 = f_l - f_k + \epsilon$, as the $\epsilon = \Delta f$ is the introduced frequency shift. Introducing $\epsilon = q\Delta\alpha$ ($\Delta\alpha = f_s/P$ and $q = \frac{\Delta f}{\Delta\alpha}$) to Eq. (8),

$$S^{\alpha_0}_{xy_T}(pL, f_{kl})_{\Delta t} \approx \sum_r X_T(rL, f_k) X^*_T(rL, f_l) g_d(p - r) e^{-i2\pi\epsilon r T_s}. \tag{9}$$

When we substitute $a_{kl} = f_k - f_l$, $f_0 = f_{kl} = (f_k + f_l)/2$ and $\alpha_0 = a_{kl} + q\Delta\alpha$ [7], the following is obtained

$$S^{a_{kl}+\Delta\alpha}_{x}(pL, f_{kl})_{\Delta t} = \underbrace{\sum_r \underbrace{\underbrace{X_T(rL, f_k) X^*_T(rL, f_l)}_{\text{Conjugate Multiplication}}\, g_d(p - r)}_{\text{windowed by g(n)}}\, e^{-i2\pi rq/P}}_{\text{P-point FFT}}\,. \tag{10}$$

The Xilinx FFT LogiCORE intellectual property (IP) core[12], provides four different architectures and a bit-accurate C library for simulation. It has AXI4-Stream compliant interfaces which were used to communicate with the other code in our implementation. It was used in the computation of the complex demodulates as it had several times higher performance per DSP than a pure HLS implementation.

3 FAM ERROR ANALYSIS

We now present an error analysis of FAM, using two separate models. The FAM_M1 model employs a fixed dynamic range where complex additions in FFT calculations are truncated according to Eq. 15. For FAM_M2, we grow the word-length for each addition, so these additions contribute zero quantization error. Details of the derivation of the output noise or signal variance of the FAM method is listed in Appendix A.

3.1 Quantization Models

3.1.1 FAM_M1 - Fixed Precision. In the FAM_M1 model, we represent signals as B-bit two's complement fractions

$$a = -a_{B-1} + \sum_{i=0}^{B-2} a_i, 2^{i-(B-1)} \tag{11}$$

where $\forall i, a_i \in \{0, 1\}$, and range $a \in [-1, 1)$. In this representation, the most significant bit determines the sign and the remaining $F = B - 1$ bits are used for the fraction.

Oppenheim et al. [16] and Widrow et al. [23] have developed a statistical quantization analysis, illustrated in Fig. 3. All operations where a signal x is quantized to x' by quantizer Q are modelled by an additive noise source n, which is assumed to be uncorrelated, uniformly distributed $\in [-2^{-F-1}, 2^{-F-1})$ and with variance $\sigma^2 = \frac{(2^{-F})^2}{12}$.

For the analysis in this paper, the computations only involve a series of multiplication and addition operations. Therefore, it is necessary to account for the quantization errors introduced by these blocks in order to deduce a quantitative analysis for the whole system. For two's complement fractions, the number of fraction bits for the product result is $2F$ which is rounded to F bits to match the wordlength. In this case, the quantization error is $\frac{(2^{-F})^2}{12} - \frac{(2^{-2F})^2}{12}$ approximated as $\sigma^2_m = \frac{(2^{-F})^2}{12}$.

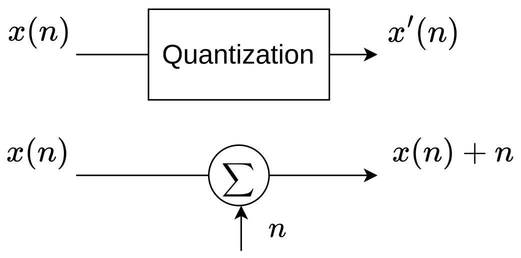


Fig. 3. Quantization noise model ($x'(n) = x(n) + n$)

Moreover, for the multiplication of two complex values, four real multiplications contribute to the variance [16, 22]:

$$\sigma_{cc}^2 = 4\sigma_m^2 = \frac{(2^{-F})^2}{3}. \tag{12}$$

For a real number multiplied with a complex number, it is

$$\sigma_{rc}^2 = 2\sigma_m^2 = \frac{(2^{-F})^2}{6}. \tag{13}$$

Quantization errors are also introduced by addition. An F-bit addition in general has an $F + 1$ bit result with no quantization error. To scale this back to F-bits without overflow requires a right shift which introduces a noise term with variance [16, 23]

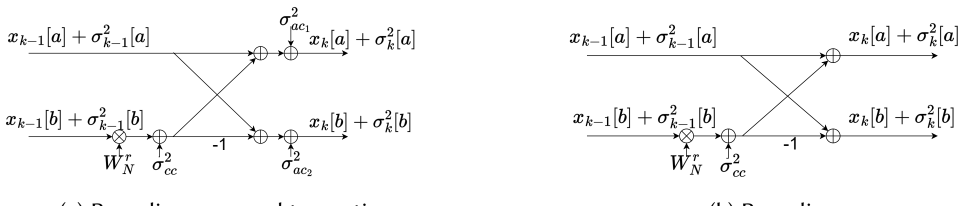
$$\sigma_{ad}^2 = \frac{(2^{-F})^2}{8}. \tag{14}$$

As the addition of two complex numbers computes the real and imaginary components separately, the variance [16, 23] becomes:

$$\sigma_{ac}^2 = 2\sigma_{ad}^2 = \frac{(2^{-F})^2}{4}. \tag{15}$$

Eq. (12) to Eq. (15) form the necessary equations for our quantization analysis.

The Fast Fourier Transform (FFT) [3] employs a butterfly structure and Fig. 4a illustrates the noise model of the Decimation-in-time (DIT) Radix-2 FFT, where N is the number of points of the FFT. This involves $m = \log_2 N$ stages. The noise model introduces round off (σ_{cc}^2) and truncation (σ_{ac}^2) terms to the system. For the kth ($1 < k < m$) stage, the input is $x_{k-1} + \sigma_{k-1}^2$ and the output is $x_k + \sigma_k^2$, with $[a]$ and $[b]$ denoting the upper and lower part of the butterfly structure.



(a) Rounding error and truncation error (b) Rounding error

Fig. 4. Quantization Noise Model of Radix-2 DIT Butterfly Structure (k^{th} Stage, 1<k<m)

To avoid overflow, a right-shift is introduced in each stage. Thus, the variance of signal and noise have to be multiplied by 1/4 before introducing the truncation noise term. Therefore, the variance is

$$\sigma_k^2[a] = (\sigma_{k-1}^2[a] + \sigma_{k-1}^2[b] + \sigma_{cc}^2)\frac{1}{4} + \sigma_{ac1}^2 \tag{16a}$$

$$\sigma_k^2[b] = (\sigma_{k-1}^2[a] + \sigma_{k-1}^2[b] + \sigma_{cc}^2)\frac{1}{4} + \sigma_{ac2}^2. \tag{16b}$$

In summary, the quantization noise arising from the FFT operation is modelled as the sum of the round off (σ_R^2) and truncation (σ_T^2) components. As $N = 2^m$ the variance of those two noise terms generated from each butterfly and passed to the final stage are

$$\begin{aligned}\sigma_R^2 &= \sigma_{cc}^2(\frac{N}{2})(\frac{1}{N})[N(\frac{1}{4})^m + \frac{N}{2}(\frac{1}{4})^{m-1} + ... + \frac{N}{2^{m-1}}(\frac{1}{4})] \\ &= \frac{1}{2}\sigma_r^2[1 - (\frac{1}{2})^m] \\ &= \frac{1}{6}2^{-2F}[1 - (\frac{1}{2})^m]\end{aligned} \tag{17}$$

and

$$\begin{aligned}\sigma_T^2 &= \sigma_{ac}^2[\frac{N}{2}(\frac{1}{4})^{m-1} + \frac{N}{4}(\frac{1}{4})^{m-2} + ... + \frac{N}{2^m}(\frac{1}{4})^{m-m}] \\ &= 2\sigma_{ac}^2[1 - (\frac{1}{2})^m] \\ &= \frac{1}{2}2^{-2F}[1 - (\frac{1}{2})^m].\end{aligned} \tag{18}$$

Thus, the output noise variance is

$$\sigma_{E\_M1}^2 = \sigma_T^2 + \sigma_R^2 = \frac{2}{3}2^{-2F}[1 - (\frac{1}{2})^m] \tag{19}$$

However, when $w \in \{1, -1, j, -j\}$, the error from multiplication will be zero. Therefore, the first two stages of DIT do not introduce noise. After extracting those from the Equation (19), the final expression of FFT quantization noise becomes

$$\sigma_{F\_M1}^2 = \sigma_{E\_M1}^2 - \sigma_{\text{stage 1 and 2}}^2 = \frac{2^{-2F}}{3}[2 - \frac{m + 1.5}{N}]. \tag{20}$$

Furthermore, based on Equation (16), the gain of this block is $\frac{1}{N}$.

In conjugate multiplication, the complex signal is multiplied by the conjugate of each thread signal, which generates quantization noise when truncating to the specified wordlength. W. Schlecker et al. derive the quantization error for the multiplication result of two quantized signals (in real number) [18] as

$$SNR = \frac{\sigma_{x1}^2\sigma_{x2}^2}{\sigma_{x1}^2\sigma_{e2}^2 + \sigma_{x2}^2\sigma_{e1}^2 + \sigma_{e1}^2\sigma_{e2}^2 + \sigma_{eq}^2} \tag{21}$$

where σ_x^2, σ_e^2 and σ_{eq}^2 is the variance of input signal, input error and the error generated by multiplication.

The error analysis assumes the signals are all independent. Hence, the signal power after complex conjugate multiplication becomes

$$P_{S.CM}^2 = P_{s1}^2P_{s2}^2 \tag{22}$$

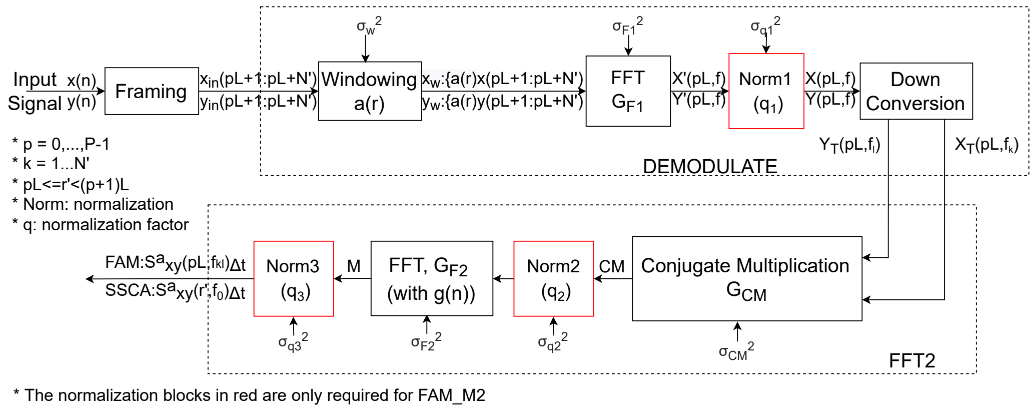


Fig. 5. SCD Signal Flow Graph for FAM_M1 (Fixed Precision) in black and FAM_M2 (Mixed Precision) techniques

and the variance of the output noise becomes

$$\sigma^2_{N.CM} = P^2_{s1}\sigma^2_{s2} + P^2_{s2}\sigma^2_{s1} + \sigma^2_{s1}\sigma^2_{s2} + \sigma^2_{CM} \tag{23}$$

where σ^2_{s1} and σ^2_2 denote the input noise sources, P_{s1} and P_{s2} are the power of two input signals, and $\sigma^2_{CM} = \sigma^2_{cc}$ is the quantization error introduced by the complex multiplication.

3.1.2 FAM_M2 - Mixed Precision. Since quantization noise is determined by the number of fractional bits, each right shift in the FAM_M1 model degrades the SQNR. Therefore, we introduce a new FAM_M2 model, which uses mixed precision to improve the SQNR by increasing the number of bits per addition and rescaling to avoid overflow [9].

Fig. 4b illustrates the new noise model for Decimation-in-time (DIT) Radix-2 FFT which has only round off noise, accompanied by a normalization to adjust the weight of the signal after the first FFT, conjugate multiplication and second FFT. The other settings are the same as in FAM_M1.

To compute the new noise model for the FFT, the variance for both lower and upper parts of the k_{th} butterfly become

$$\sigma^2_k[a] = \sigma^2_{k-1}[a] + \sigma^2_{k-1}[b] + \sigma^2_{cc} \tag{24a}$$

$$\sigma^2_k[b] = \sigma^2_{k-1}[a] + \sigma^2_{k-1}[b] + \sigma^2_{cc}. \tag{24b}$$

In the k_{th} stage, $N_c/2$ points are multiplied with twiddle factors, introducing an error that doubles when passing through each of the $m-k+1$ subsequent stages. Thus, the variance of output of the FFT can be computed from Eq. (24) to

$$\begin{aligned}\sigma^2_{E\_M2} &= \sigma^2_{cc}\frac{N_c}{2}\frac{1}{N_c}[2^m + 2^{m-1} + ... + 2] \\ &= (2^m - 1)\sigma^2_{cc}.\end{aligned} \tag{25}$$

Removing the quantization error of the twiddle factor $\in \{i, -i, 1, -1\}$ from the Eq. (25), the variational expression for the FFT quantization error is

$$\sigma^2_{F\_M2} = [\frac{2^m}{6} - 1]\frac{2^{-2F}}{3}. \tag{26}$$

Table 1. Summary of gain and quantization error for each block of FAM_M1 and FAM_M2

Blocks		Gain	Block Error	Output Error	Output Signal	Integer bits
FAM_M1 FAM_M2	Framing	1	-	-	P_s	1
	Windowing	$G_W = \frac{1}{1.59^2}$ [4]	$\sigma_w^2 = \sigma_{rc}^2$	$\sigma_{N.W}^2 = \sigma_{rc}^2$	$P_{S.W} = \frac{P_s}{1.59^2}$	1
	Down Conversion	1	-	-	-	1
FAM_M1	First FFT	$G_{F_1} = 1/N_c$	$\sigma_{F1}^2 = \sigma_{F\_M1}^2$	$\sigma_{N.F1}^2 = \sigma_{N.W}^2/N_c + \sigma_{F1}^2$	$P_{S.F1} = P_{S.W}/N_c$	1
	Conjugate Multiply	G_{CM}	$\sigma_{CM}^2 = \sigma_{cc}^2$	$\sigma_{N.CM}^2$	$P_{S.CM}$	1
	Second FFT	$G_{F_2} = 1/N_f$	$\sigma_{F2}^2 = \sigma_{F\_M1}^2$	$\sigma_{N.F2}^2 = \sigma_{N.W}^2/N_f + \sigma_{F2}^2$	$P_{S.F2} = P_{S.CM}/N_f$	1
FAM_M2	First FFT	$G_{F_1} = N_c$	$\sigma_{F1}^2 = \sigma_{F\_M2}^2$	$\sigma_{N.F1}^2 = \sigma_{N.W}^2 N_c + \sigma_{F1}^2$	$P_{S.F1} = P_{S.W} N_c$	$log_2 N_c + 1$
	Normalization 1	$G_{Norm_1} = q_1^2$	$\sigma_{q1}^2 = \sigma_{rc}^2$	$\sigma_{N.q1}^2 = \sigma_{N.F1}^2 q_1^2 + \sigma_{q1}^2$	$P_{S.q1} = P_{S.F1} q_1^2$	1
	Conjugate Multiply	G_{CM}	$\sigma_{CM}^2 = \sigma_{cc}^2$	$\sigma_{N.CM}^2$	$P_{S.CM}$	2
	Normalization 2	$G_{Norm_2} = q_2^2$	$\sigma_{q2}^2 = \sigma_{rc}^2$	$\sigma_{N.q2}^2 = \sigma_{N.CM}^2 q_2^2 + \sigma_{q2}^2$	$P_{S.q2} = P_{S.CM} q_2^2$	1
	Second FFT	$G_{F_2} = N_f$	$\sigma_{F2}^2 = \sigma_{F\_M2}^2$	$\sigma_{N.F2}^2 = \sigma_{N.q2}^2 N_f + \sigma_{F2}^2$	$P_{S.F2} = P_{S.q2} N_f$	$log_2 N_f + 1$
	Normalization 3	$G_{Norm_3} = q_3^2$	$\sigma_{q3}^2 = \sigma_{rc}^2$	$\sigma_{N.q3}^2 = \sigma_{N.F2}^2 q_3^2 + \sigma_{q3}^2$	$P_{S.q3} = P_{S.F2} q_3^2$	1

3.2 FAM Signal Flow Diagram

Fig. 5 illustrates the signal flow diagram for FAM_M1 and FAM_M2. The gain and variance introduced by each block in Fig. 5 are calculated using the equations from the previous subsection and summarized in Table 1. In the figure, the noise introduced in each block is the block error (σ_*^2), the output error ($\sigma_{N.*}^2$) is the sum of the block error and the output error from previous block multiplied by the gain of the current block, and the power of the signal passing out of each block is the output signal ($P_{S.*}$) where $*$ refers to a particular block in the signal flow.

3.2.1 Framing + Windowing. The Framing block rearranges the data sequence $x(n)$ into P segments, defined as $x(pL+1 : pL+N_c)$ where $p = 0...P-1$. For the Windowing block, $a(N_c)x(pL+1 : pL+N_c)$ in Eq. (6) shows the multiplication between the inputs and the Hamming Window yielding a rounding error ($\sigma_w^2 = \sigma_{rc}^2$). In addition, the power correlation factor of the Hamming Window is 1.59 ($\approx 1/(RMS(Hamming\,Window))$) where RMS is root-mean-square. So, there is approximately a 4 dB ($\approx 10log_{10}(1.59^2)$) reduction in SQNR after windowing.

3.2.2 The First Fast Fourier Transform. DIT FFT blocks are employed to implement Eq. (6). The noise power is given by Eq. (20) where N_c is the points of FFT, m_1 is the stages of FFT. The gain of this block is $1/N_c$.

3.2.3 Down Conversion. To implement the complex demodulate, the FFT output needs to be down converted. To control cycle leakage and aliasing, L is set to $N_c/4$ and therefore $e^{-i2\pi mpL/N_c}$ can only take the values (i, -i, 1, -1) and does not introduce quantization error.

3.2.4 Conjugate Multiplication. The function of this block is to compute a complex dot product in Eq. (10) and the underlying computation is to multiply an input with its conjugate. We assume those two signals are independent but have the same input signal and noise power ($P_{S.S} = P_{s1} = P_{s2}$ and $\sigma_{N.S}^2 = \sigma_{s1}^2 = \sigma_{s2}^2$). Thus, Eq. (23) can be simplified to $\sigma_{N.CM}^2 = 2P_{S.S}^2\sigma_{N.S}^2 + \sigma_{N.S}^4 + \sigma_{CM}^2$ where $\sigma_{N.S}^4$ is approximated as zero due to its high order. The power of the output signal, $P_{S.CM}$, is $P_{S.S}^2$.

3.2.5 The Second Fast Fourier Transform. The second FFT is an N_f point one, where $N_f = P$. Similar to the First FFT, the noise power of the Second FFT is given by Eq. (20) where N_f is FFT size and $m_2 = log_2(N_f)$ is the number of stages. The gain of this block is $1/N_f$.

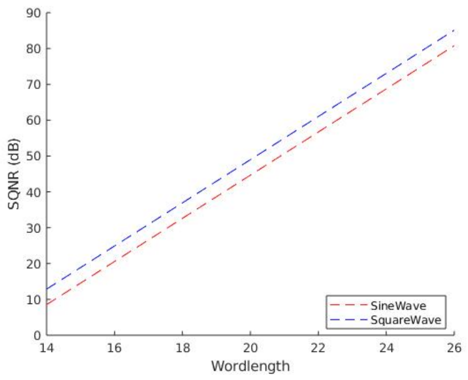


Fig. 6. SQNR performance for the FAM_M1 Method at different wordlengths

3.3 SQNR Noise Model for FAM_M1

The FAM_M1 model employs a fixed wordlength for all signals. This is achieved by truncating multiplications and right shifting additions so that the result remains in the range $[-1, 1)$.

Combining all the noise terms in Fig. 5, the output noise variance (σ^2) of FAM_M1 can be reduced to the following form

$$\begin{aligned}\sigma^2_{FAM\_M1} &= [(\sigma^2_W G_{F_1} + \sigma^2_{F_1})G_{CM} + \sigma^2_{CM}]G_{F_2} + \sigma^2_{F_2} \\ &= W_W 2^{-2F_W} + W_{F_1}2^{-2F_1} + W_{CM}2^{-2F_{CM}} + W_{F_2}2^{-2F_2} \\ &= \sum_{\#\in\{W,F_1,CM,F_2\}} W_\# 2^{-2F_\#}\end{aligned} \tag{27}$$

where $W_\#$ and $F_\#$ are the parameters and the number of fraction bits of each section. The power of output signal is

$$P_{FAM\_M1} = P_{i.s}G_W G_{F_1} G_{CM} G_{F_2} \tag{28}$$

where $P_{i.s}$ is the power of the input signal. The detailed derivations for the FAM_M1 is given in Appendix A.1.

The SQNR for the FAM_M1 method is

$$SQNR = 10\log_{10}(\frac{P_{FAM\_M1}}{\sigma^2_{FAM\_M1}}). \tag{29}$$

Fig. 6 shows the SQNR analysis using Eq. (29) for both sine wave and square wave input using the FAM_M1. A low SQNR results from scaling of each addition operation through a right shift (division by 2) which is necessary to avoid overflow.

3.4 SQNR Noise Model for FAM_M2

The FAM_M2 model increases the number of bits for each addition and rescaling to avoid overflow. As a result, this design is a mixed precision one and only rounding errors are introduced. Since overflow cannot occur, in either FFT or the conjugate multiplications, the Framing, Windowing and Down Conversion blocks remain unchanged from FAM_M1.

3.4.1 The First Fast Fourier Transform + Normalization 1. The noise power of the new system is given by Eq. (26). Because additions in each FFT stage require an additional integer bit to avoid overflow, the complex output is scaled via the (real) normalization factor q_1 so the real and imaginary parts are $\in [-1, 1)$. This introduces variance σ_{q1}^2.

3.4.2 Conjugate Multiplication + Normalization 2. The computation part is same as the previous method. The normalized factor for this part is q_2, which rescales to produce a variance σ_{q2}^2.

3.4.3 The Second Fast Fourier Transform + Normalization 3. The second FFT is N_f points to estimate the sum with q_3 normalization factor and variance σ_{q3}^2.

3.5 FAM_M2 - SQNR Calculation

Combining all the noise and gain values of Fig. 5 and using the definitions in Table. 1, the output noise variance and power of output signal can be written as

$$\begin{aligned}\sigma_{FAM\_M2}^2 &= [((((\sigma_W^2 G_{F_1} + \sigma_{F_1}^2)G_{Norm_1} + \sigma_{q_1}^2)G_{CM} + \sigma_{CM}^2)G_{Norm_2} + \sigma_{q_2}^2)G_{F_2} + \sigma_{F_2}^2]G_{Norm_3} + \sigma_{q_3}^2\\ &= W_W 2^{-2F_W} + W_{F_1}2^{-2F_1} + W_{CM}2^{-2F_{CM}} + W_{F_2}2^{-2F_2}\\ &= \sum_{\#\in\{W,F1,CM,F2\}} W_\# 2^{-2F_\#}\end{aligned} \tag{30}$$

and

$$P_{FAM\_M2} = P_{i.s} G_W G_{F_1} q_1^2 G_{CM} q_2^2 G_{F_2} q_3^2. \tag{31}$$

The detailed derivations for the FAM_M2 is given in Appendix A.2.

4 IMPLEMENTATION

The FAM algorithm can be decomposed into three sections, DEMODULATE, FFT2, and Sparse SCD. Fig. 5 indicates the details of the first two parts, and the Sparse SCD will be presented in Section 4.3. Beginning with a naive baseline implementation, we propose a second design where computational efficiency is maximised through parallelism. Finally, a technique for I/O bandwidth reduction which improves system-level performance is described. We design an HLS-based parallel architecture and integrate it into a Zynq processing system on the Xilinx ZCU111 board, on top of which a Jupyter notebook is developed to visualize the results.

C/C++-based synthesis via Vivado HLS was chosen in preference to a register transfer language (RTL) design flow such as VHDL or Verilog because it allows non-FPGA experts to modify the code, directly supports fixed-point types, and has high design productivity.

The ap_fixed [11] bit-accurate fixed-point library, greatly facilitated comparison of our theoretical SQNR models with simulations, and hardware implementations in Verilog could be generated from the same source code. In this library a fixed-point data type is represented using the C++ template $ap_[u]$fixed $< W, I, Q, O, N >$, where W is the wordlength in bits, I the number of integer bits, Q the quantization mode, O the overflow mode and N is the number of saturation bits in the overflow wrap mode [11].

4.1 Baseline Implementation

An implementation based on Fig. 5 was first developed. The system accepts a window of input data and calculates all the outputs for each block, and the results are streamed to subsequent blocks. Parallelism between blocks is achieved via DATAFLOW pragmas in the HLS description. Pseudocode for the baseline implementation is listed in Algorithm 1. The blocks operate independently in a pipelined manner so the throughput is equal to the throughput of the block with highest initiation interval (II). The Second FFT part requires $\mathbb{O}(N_c^2)$ operations and hence is the computational bottleneck.

Algorithm 1: Baseline Implementation.

```
Input: Stream in.
Output: Stream out.
# pragma HLS DATAFLOW
// Framing
$xin[0 : P - 1, 0 : N_c - 1] \leftarrow Framing(in.read())$
// Windowing
$x_w[0 : P - 1, :] \leftarrow xin[0 : P - 1, :] * a[:]$
// First FFT
$X[0 : P - 1, :] \leftarrow FFT(x_w[0 : P - 1, :])$ //$N_c$ point FFT
// Down Conversion
$X_T[0 : P - 1, :] \leftarrow DownConversion(X[0 : P - 1, :])$
// Conjugate Multiplication
for $i \leftarrow 0$ to $N_c - 1$ by 1 do
    for $j \leftarrow 0$ to $N_c - 1$ by 1 do
        $CM[:, i * N_c + j] \leftarrow X_T[:, i] * conj(X_T[:, j])$
    end
end
//Second FFT
$M[:, 0 : N_c * N_c - 1] \leftarrow FFT(CM[:, 0 : N_c * N_c - 1])$ //$N_f$ point FFT
$P_a[:, :] \leftarrow M[P/2 : (3P/4 - 1), 0 : N_c * N_c - 1]$
$P_b[:, :] \leftarrow M[P/4 : (P/2 - 1), 0 : N_c * N_c - 1]$
$out.write() \leftarrow \{P_a, P_b\}$
return $out$
```

4.2 Computation Optimization

To optimize the design we employ spatial parallelism by instantiating DSTRIDE parallel units for the DEMODULATE computation and FSTRIDE parallel units for FFT2. We note that DEMODULATE (framing, windowing, N_c-point FFT and down-conversion) is computed row-wise, whereas FFT2 requires column-wise inputs (Algorithm 1). This makes their boundary a natural place for a pipeline stage.

Our overall strategy is to create a pipeline where the DEMODULATE and FFT2 stages have a similar II. In Algorithm 2, we merge related blocks in the same loop, so that we can circulate arrays with N_c items in complex demodulation, or pass arrays with P items in the second part of the loop. Streams are characterized by reading or writing once in each loop, and the HLS PARTITION pragma is used to parallelise array accesses.

The pseudo-code for computing the DEMODULATE and FFT2 stages are presented as Algorithm 3. Referring to Fig 5, the x_w, X, and X_T variables represent the output arrays of windowing, first FFT, and down conversion steps (described in Section 3.2) respectively. We implement the FFT for the DEMODULATE block using the Xilinx FFT library. The FFT IP core library computes the unscaled fixed point precision DIT FFT and rounds to the specified wordlength after the butterfly, which means the rounding error comes from real and imaginary part of the complex result.

Algorithm 2: Merging of Algorithm 1.

```
Input: Stream in.
Output: Stream out.
# pragma HLS DATAFLOW
// Framing
$xin[0 : P - 1, 0 : N_c - 1] \leftarrow Framing(in.read())$
//DEMODULATE
for $i \leftarrow 0$ to $P - 1$ by $1$ do
    // Windowing
    $x_w[:] \leftarrow xin[i, :] * a[:]$
    // First FFT
    $X[:] \leftarrow FFT(x_w[:])$ //$N_c$ point FFT
    // Down Conversion
    $X_T[i, :] \leftarrow DownConversion(X[:])$
end
// FFT2
for $i \leftarrow 0$ to $N_c - 1$ by $1$ do
    for $j \leftarrow 0$ to $N_c - 1$ by $1$ do
        $CM[:, i * N_c + j] \leftarrow X_T[:, i] * conj(X_T[:, j])$
        $M[:] \leftarrow FFT(CM[:, i * N_c + j])$ //$N_f$ point FFT
        $P_a[:] \leftarrow M[P/2 : (3P/4 - 1)]$
        $P_b[:] \leftarrow M[P/4 : (P/2 - 1)]$
        $out.write() \leftarrow \{P_a, P_b\}$
    end
end
return out
```

The DEMODULATE computation requires less resources than the following FFT2 part and so compile-time parameters are introduced to balance the II of each stage. In Algorithm 3, DSTRIDE controls the degree of parallelism. The *save_in* function is used for buffering. The final stage of DEMODULATE, *array_reorder*, is transferring the matrix from the size of $DSTRIDE \times (P/DSTRIDE) \times N_C$ to $N_c \times P$ and reordering in preparation for FFT2.

Fig. 7 illustrates the data flow for the FFT2 part of the FAM implementation. The initial stages prepare data for parallel Conjugate Multiplication + FFT (CMF) units (Fig. 8). FSTRIDE determines the degree of parallelism used in the CMF units. A total of FSTRIDE×CMF units are operated in parallel, with each CMF unit optimised for minimal II. Together this has $execution\,time = N_c^2 II_{CMF}/FSTRIDE$. For high performance we wish to have $II_{CMF} = 1$. This can be achieved under the following conditions represented in Fig. 8:

- an array and its conjugate should be passed to the CMF unit each cycle;
- arrays must use PARTITION to read or write values each cycle;
- the inner for loops must be UNROLLed;
- the computation of each stage and the buffers for the result in the N_f-point FFT must all be independent;
- the N_f-point FFT executes all butterflies in parallel.

Algorithm 3: Optimization through spatial parallelism.

```
Input: Stream in.
Output: An array of stream $Out$ with a size of $FSTRIDE \times P/2$.
# pragma HLS DATAFLOW
// DEMODULATE
$xin[:][:,:] \leftarrow save\_in(in.read())$ // The size of $xin$ is $[DSTRIDE][P/DSTRIDE, N_c]$
for $i \leftarrow 0$ to $P/DSTRIDE - 1$ do
    for $n \leftarrow 0$ to $DSTRIDE - 1$ do
        # pragma HLS UNROLL
        $Preprocess$: // Preprocess is a function with following sub_functions
        {
        # pragma HLS DATAFLOW
        $x_w[n][i,:] \leftarrow xin[n][i,:] * a[:]$
        $X[n][i,:] \leftarrow FFT(x_w[n][i,:])$ //$N_c$ point FFT
        $Y[n][i,:] \leftarrow DownConversion(X[n][i,:])$
        }
    end
end
$X_T[:, 0 : P-1] \leftarrow array\_reorder(Y[0 : DSTRIDE-1][0 : P/DSTRIDE-1,:])$
// FFT2
for $n \leftarrow 1$ to $FSTRIDE$ do
    $\#pragma\, HLS\, ARRAY\_PARTITION\, variable = X_{n.conj}\, complete\, dim = 2$
    $X_{n.conj}[:,:] \leftarrow Conjugate(X_T[n : FSTRIDE : N_c - 1, :])$
end
for $i \leftarrow 0$ to $N_c - 1$ by 1 do
    $X_{temp}[:] \leftarrow X_T[i, :]$
    for $j \leftarrow 0$ to $N_c - 1$ by $FSTRIDE$ do
        # pragma HLS PIPELINE = 1
        for $n \leftarrow 0$ to $FSTRIDE - 1$ by 1 do
            # pragma HLS UNROLL
            CMF unit:
            {
            $CM[:] \leftarrow X_{temp}[:] * X_{n.conj}[j+n, :]$
            $M[:] \leftarrow FFT(CM[:])$ //$N_f$ point FFT
            $P_a[:] \leftarrow M[P/2 : (3P/4 - 1)]$
            $P_b[:] \leftarrow M[P/4 : (P/2 - 1)]$
            $Out[n * (P/2) : (n+1) * (P/2)].write() \leftarrow \{P_a[:], P_b[:]\}$
            }
        end
    end
end
return $stream\, Out[0 : FSTRIDE * P/2]$
```

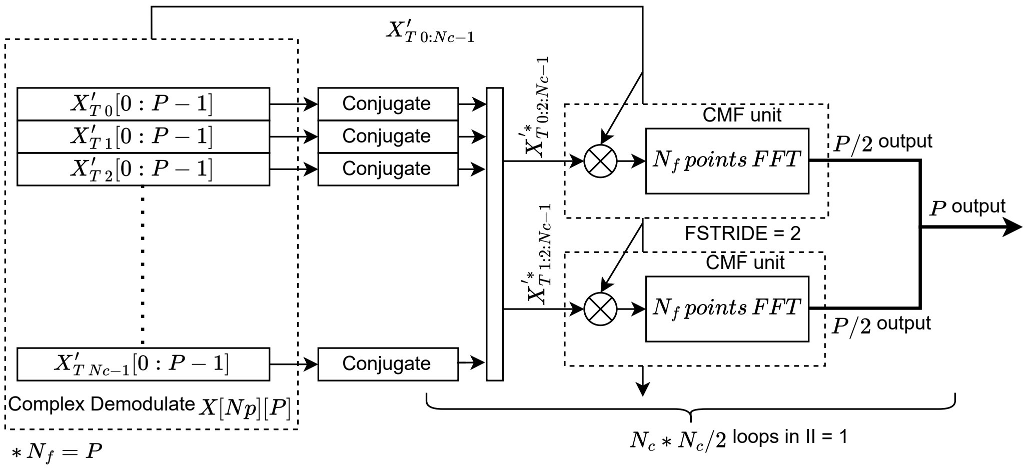


Fig. 7. SCD Signal Flow Graph (CM+FFT) Based On HLS Design with FSTRIDE=2 (Part 2)

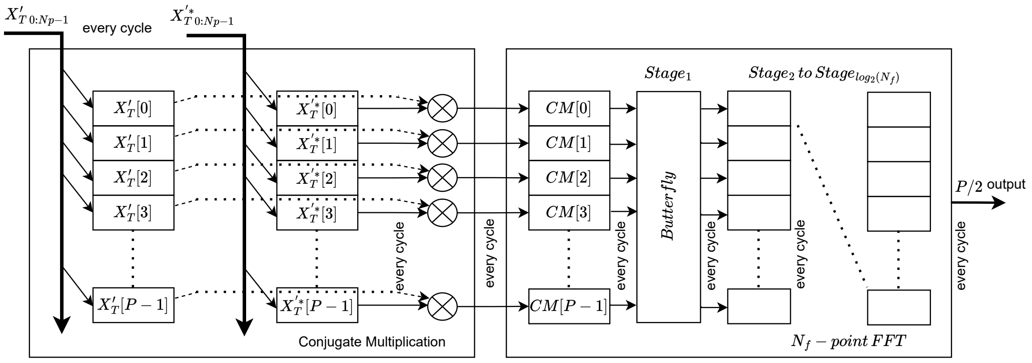


Fig. 8. CMF unit data flow

The FFT2 part of Algorithm 3 shows the pseudo-code for Fig. 7. Before passing the data to the CMF unit, we split and save the conjugate values into FSTRIDE matrices of size $N_c/FSTRIDE \times P$ and PARTITION as $dim = 2$ (the second dimension is partitioned). The data is then transferred and run into the FSTRIDE× CMF units synchronously. To accept coming data set in the next cycle, the CMF is fully expanded by UNROLL in a for loop to compute the complex multiplication and the result is stored in a new CM array. This is then passed to the N_f-point FFT, and similarly, after each stage of the FFT, the results are saved and passed in a new array while all the butterfly functions in the for loop are fully expanded and computed.

The FFT2 requires a $N_f log_2(N_f)/2$ butterfly operations and, as illustrated in Fig. 4b, requires one complex multiplication (two real multiplications and four real additions) and two complex additions (four real additions). In FFT2, executing all butterflies in parallel will cause high computational complexity. Thus, for the twiddle factor whose value is $(1,\ -1,\ j,\ -j)$, the butterfly calculation can be done by addition only. Therefore, we replace the first two stages of the FFT with pure addition, so that the complex multiplication of $2/log2(N_f)$ can be reduced.

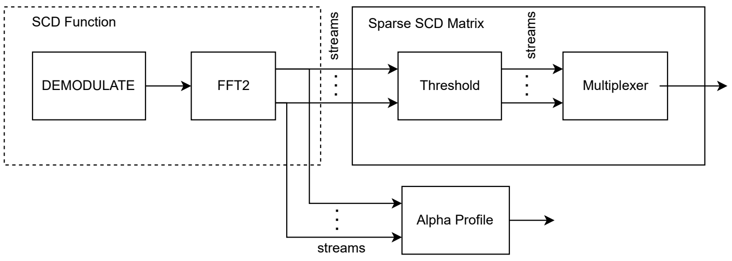


Fig. 9. The SCD function is reduced either via thresholding or computing the alpha profile.

Table 2. Parameters chosen for our design. We set $L = N_c/4$ and $N_f = P = N/L$.

Parameters	N	N_c	P	N_f	L	$DSTRIDE$	$FSTRIDE$
value	2,048	256	32	32	64	1 or 4	2

4.3 I/O Optimization

As illustrated in Fig. 9, our architecture first computes the entire SCD matrix (dotted box) using the approach just described. The number of streams in red in Fig. 9 is $FSTRIDE \times P/2$. The result is a dense matrix, but we are only interested in entries with high correlation, those values being less than 1% of the complete matrix. The right-hand box illustrates a block to create a sparse SCD matrix. It thresholds multiple, parallel streams and then multiplexes them into a single one. The bottom box also illustrates the alpha profile which we have not implemented, but could be easily included. In Eq. 3, the alpha profile is the maximum amplitude of the SCD matrix, mapped to the alpha frequency axis [2, 13, 14]. The sparse SCD matrix provides an additional dimension of information over the alpha profile.

Returning a sparse matrix or the alpha profile both serve to reduce the amount of data transferred from accelerator to the host. This is fortunate because the SCD matrix has dimension $2N \times 2N_c$, e.g. for the example of Fig. 1c, value of N_c and N are set in Table 2, the output has 524,288 values. In comparison, the sparse matrix fomat in Fig. 1d has only 2,396 values but is able to capture the features of interest.

Our implementation of the FAM method just described outputs $FSTRIDE \times P/2$ parallel streams, each having a width of $N_c^2/FSTRIDE$ values. As the number of CMF cells increases, the number of output streams increases, which requires more I/O transactions. Algorithm 4 describes how we filter and output the target value with associated position information and eventually merge the multiple data streams to a single one. After thresholding, a converter function combines the nonzero value and its coordinates into an Int64 data type which is streamed to the Multiplex function, which combines data from the parallel streams to produce a stream of outputs that includes value and the label information (frequency label (flabel) and alpha frequency label (alabel)).

4.4 Exploiting Symmetry

In Section 2 we have shown that the representation of the SCD estimation $\hat{S}_x^\alpha(f)$ is a 2-D feature map with the f and α axes. It is symmetrical in the bi-frequency dimension [6] as indicated in Equations (32) and (33).

[1] Algorithm 4: for $FSTRIDE = 2$;

Algorithm 4: Sparse SCD

```
// Threshold
Input: FSTRIDE array stream $In_{1,\ldots,FSTRIDE}$, each of size of $P/2$ and sequence order $countn$.
Output: FSTRIDE array stream $Out_{1,\ldots,FSTRIDE}$.
for $j \leftarrow 1$ to $FSTRIDE$ by 1 do
    # pragma HLS UNROLL
    for $i \leftarrow 0$ to $P/2-1$ by 1 do
        # pragma HLS UNROLL
        if $In_j[i] > THRESHOLD$ then
            $alabel \leftarrow alpha\_label(countn)$
            $flabel \leftarrow frequency\_label(countn)$
            $Out_j[i] \leftarrow (int64)pack\{In_j[i], alabel, flabel\}$
        end
        countn = countn+1
    end
end
return $stream\ (Out_{1,\ldots,FSTRIDE}[0:P/2])$
// Multiplex
Input: FSTRIDE array stream $In_{1,\ldots,FSTRIDE}$, each of size $P/2$.
Output: Three streams $value$, $alabel$ and $flabel$.
for $j \leftarrow 1$ to $FSTRIDE$ by 1 do
    for $i \leftarrow 0$ to $P/2-1$ by 1 do
        $\{value, alabel, flabel\} \leftarrow unpack(In_j[i])$
    end
end
return $stream\ (value, alabel, flabel)$
```

$$\hat{S}_x^{\alpha}(f) = \hat{S}_x^{\alpha}(-f) \tag{32}$$

$$\hat{S}_x^{-\alpha}(f) = \hat{S}_x^{\alpha}(f)^* \tag{33}$$

Thus, it is sufficient to compute a quarter of the SCD matrix $\hat{S}_x^{\alpha}(f)$, reducing computation by 25%. Fig. 10 shows an example for $N_c = 256$ and $N = 2,048$ (the i and j symbols are from the FFT2 section of Algorithm 3). Note that only Pa and Pb contribute to the SCD matrix, represented by the light and dark colors of the same color. For each color in Fig. 10, different rows of X_T are represented, and for same color from left to right j increases from 1 to N_c representing the rows of the conjugate matrix, which are arranged in the SCD matrix in the staggered order shown in Fig. 10. Moreover, in Fig. 10, the shaded area is the quarter SCD, and it is obtained by changing $i \in [0:1:N_c-1]$ to $i \in [0:1:N_c/2-1]$ and $j \in [0:1:N_c-1]$ to $j \in [i:1:N_c/2-1-i]$ in Algorithm 3.

4.5 Cycle Count Summary

After applying the optimizations described above, the pipelining scheme for the DEMODULATE and FFT2 blocks is illustrated in Fig 11. Since FFT2 stage is the computational bottleneck, we design the II of FFT2 to meet throughput requirements and then ensure the II of DEMODULATE

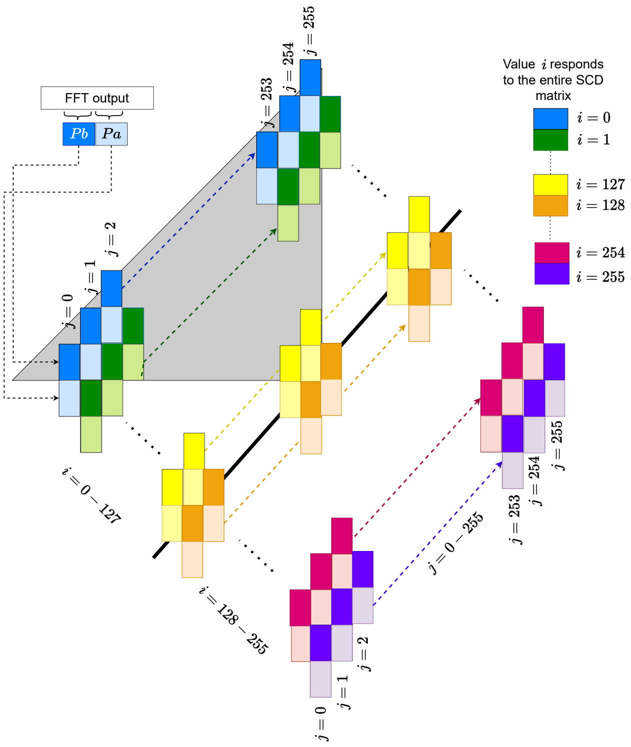


Fig. 10. An example of an SCD matrix (symbol i and j are from Algorithm 3 FFT2).

($II_{DEMODULATE}$) is less than or equal to II_{FFT2}. We apply a DATAFLOW pragma so that the II of each block is equal to the maximum II over the sub-blocks multiplied by the number of iterations. For the parameters in Table 2, the II_{FFT2} of the full and quarter SCD matrix computations are 32,768 and 8,192 respectively. From synthesis reports, the II of save_in, preprocess and save_out are 1, 875 and 4. Therefore, the IIs of the sub-blocks of DEMODULATE are $II_{save\_in} = N_c * P = 8,192$, $II_{Preprocess} = max\{II_W, II_{IP}, II_{DC}\} * P/DSTRIDE = 874 * 32/DSTRIDE = 27,968/DSTRIDE$, $II_{save\_out} = N_c * P = 8,192$. Thus to match II_{FFT2}, DSTRIDE should be set to 1 and 4 for computing the full and quarter SCD matrix.

5 RESULTS

We created the IP blocks for FAM_M1 and FAM_M2 separately using the Vivado HLS 2020.1 High-level synthesis tool. Then Vivado 2020.1 was used to generate bitstreams which were tested on a Xilinx ZCU111 RFSoC board which uses a Zynq UltraScale+ XCZU28DR-2FFVG1517E device. We verified the accuracy between the computational expressions of the two methods FAM_M1 and FAM_M2 and the actual operation, and compared a range of information regarding SQNR and resource and energy consumption. Although the ZCU111 is larger than needed for just the IP blocks, it was chosen so we could accommodate designs with larger wordlengths and more parallelism. Moreover, future work will integrate the FAM core with high-speed ADCs and deep learning. Our implementation is parameterized, and we report on a design with FAM parameters matching the literature [2, 14] which were summarized in Table 2. Our choice of parallelization parameters are $FSTRIDE = 2$ and $DSTRIDE = 1$ (Full SCD) or $DSTRIDE = 4$ (Quarter SCD).

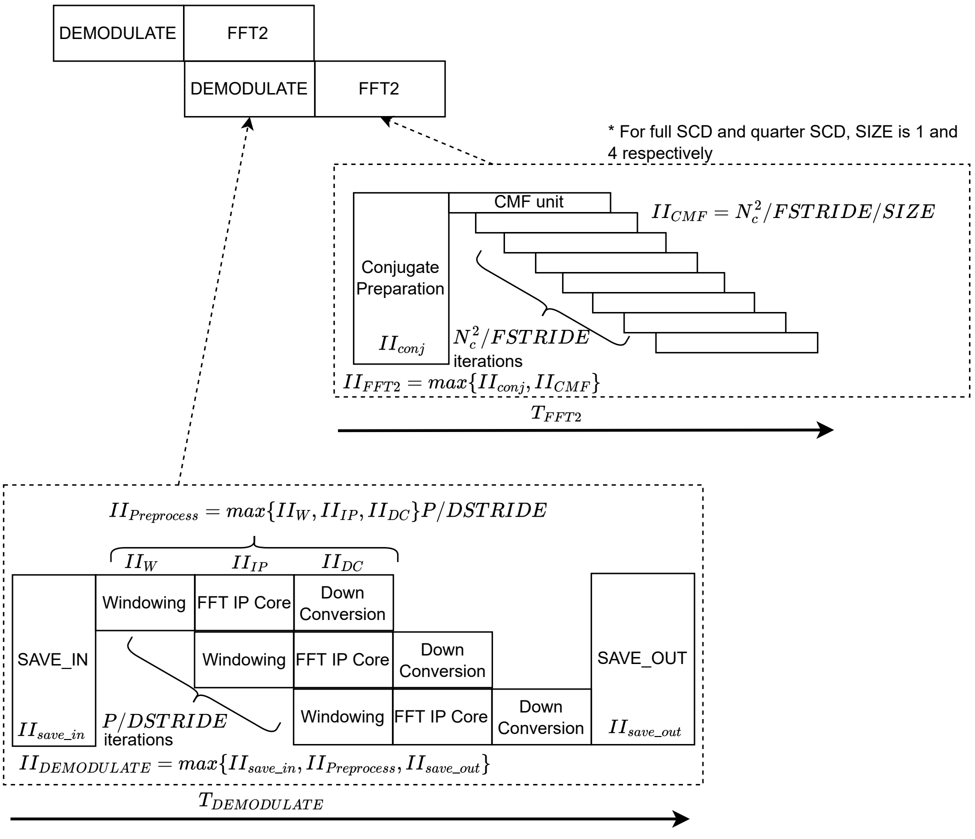

DEMODULATE
FFT2
DEMODULATE
FFT2
* For full SCD and quarter SCD, SIZE is 1 and 4 respectively
CMF unit
$II_{CMF} = N_c^2/FSTRIDE/SIZE$
Conjugate Preparation
II_{conj}
$N_c^2/FSTRIDE$ iterations
$II_{FFT2} = max\{II_{conj}, II_{CMF}\}$
T_{FFT2}
$II_{Preprocess} = max\{II_W, II_{IP}, II_{DC}\}P/DSTRIDE$
II_W
II_{IP}
II_{DC}
SAVE_IN
II_{save_in}
Windowing
FFT IP Core
Down Conversion
$P/DSTRIDE$ iterations
SAVE_OUT
II_{save_out}
$II_{DEMODULATE} = max\{II_{save\_in}, II_{Preprocess}, II_{save\_out}\}$
$T_{DEMODULATE}$

Fig. 11. A cycle-aware system flowchart with pipeline stage details.

5.1 SQNR

Bit-accurate simulations were made through a direct implementation of the FAM_M1 and FAM_M2 algorithms in C. The ap_fixed type in Vivado HLS was used for fixed point arithmetic. The simulations were compared with our mathematical derivations using sine-waves, square-waves and samples from the DeepSig RADIOML 2018.01A dataset [10]. On FPGA devices, wordlengths up to 18-bits are supported by the embedded DSP blocks, and additional bits can be implemented using the programmable logic so experiments were focused around this and higher values.

Fig. 12 shows the SQNR for different uniform wordlengths for Deepsig, Sine Wave and Square Wave signals using the different methods mentioned in Section 3. The FAM_M2 techniques have considerably improved SQNR compared with FAM_M1. For each signal and method, it can be seen in Fig 12 that there is a 6 dB improvement in the SQNR with each additional bit because $2^{-2(F-1)} = 4 \times 2^{-2F}$. The traces in Fig. 12 are input-signal dependent because Eq. (28) and Eq. (31) depend on the input power. Note that for different input signals, the normalization involves different scale factors, which affects a direct comparison with FAM_M2.

5.1.1 *SQNR vs Number of Bits.*

One of the benefits of the proposed approach is that it enables us to easily determine the best bit allocation to achieve the highest SQNR. Noting that Eq. (27) and Eq. (30) are simple sums of products and the SQNR is given by Eq. (29), the proposed approach to

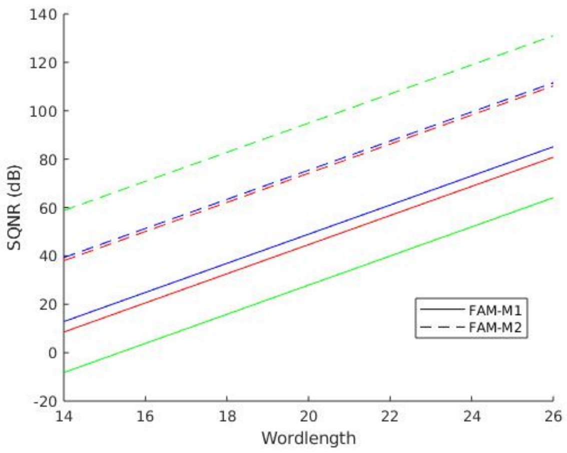


Fig. 12. SQNR performance for the FAM Method in our models FAM_M1 and FAM_M2 (theory) at different wordlengths B ($F = B - 1$) (Red: Sine Wave; Blue: Square Wave; Green: Deepsig;)

minimizing this expression results in each block providing an equal contribution to σ^2. Given the $W_\#$ values, and assigning F bits to the FFT2 stage, we make each sum term in Eq. 27 and Eq. 30 equal via the allocation in Table 3.

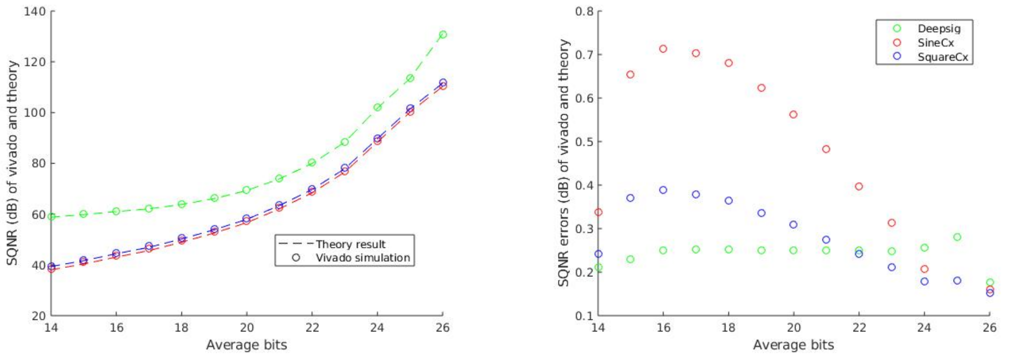
Table 3. Reduce wordlength with less impact on SQNR

	Methods	Window	FFT1	CM	FFT2
DeepSig	FAM_M1	$F-13$	$F-8$	$F-3$	F
	FAM_M2	$F-1$	$F-1$	F	F
SineWave	FAM_M1	$F-11$	$F-6$	$F-2$	F
SquareWave	FAM_M2	$F-6$	$F-4$	$F-1$	F

Table 4. Bit allocation for best SQNR using exhaustive search (F_{sum} = 72 Signal: DeepSig) vs Uniform

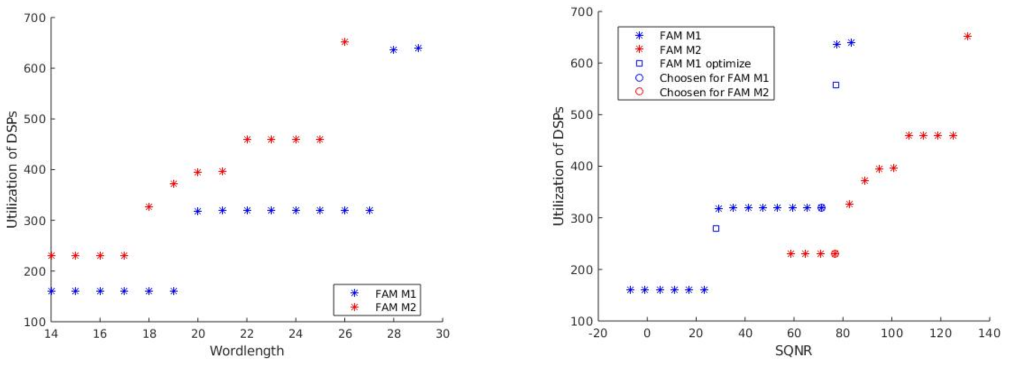
Methods	Uniform	Window	FFT1	CM	FFT2	SQNR
FAM_M1	No	11 (24-13)	16 (24-8)	21 (24-3)	24	46.04
	Yes	18	18	18	18	15.87
FAM_M2	No	17 (19-2)	17 (19-2)	19	19	84.29
	Yes	18	18	18	18	82.85

Table 4 illustrates the potential benefits of using a non-uniform wordlength throughout the computation. This example is run on the Deepsig input, with the optimal bitwidth allocation computed by an exhaustive search over all of the possible bit allocations, for a fixed number of total bits $F_{sum} = 72$. We can see that by using a non-uniform number of bits, it enables one to achieve a higher SQNR. Note that exhaustive search results have reached similar bitwidth configurations to our formulas from Table 3; the minor difference being that the formula for FAM_M2 is not designed for a maximum of $F_{sum} = 72$ total bits.



(a) The comparison of theory and Vivado simulation (b) Absolute error between theory and simulation

Fig. 13. Simulation result in average bits FAM_M2 (Red: Sine Wave; Blue: Square Wave; Green: Deepsig;)



(a) Different wordlength (b) Different SQNR

Fig. 14. FAM methods for DSP utilization at different conditions

5.2 Vivado HLS Simulation

Bit accurate simulations using Vivado_HLS were used to verify the theoretical results of Section 3.4. The designs are described in C, compiled and executed to obtain a bit-exact result. Since the theory directly calculates SQNR from the SCD parameters, it is orders of magnitude faster than simulation using HLS.

Fig. 13a shows the average SQNR with non-uniform accuracy for FAM_M2 for theoretical and HLS simulations with different input signals, where the average number of bits = $(B_W + B_1 + B_{CM} + B_2)/4$ ($B_\# \in [14, 26]$ in steps of 4). Fig. 13b shows the difference between the theoretical and Vivado simulations in Fig. 13a. The result shows our models described in Section 3.3 and Section 3.5 can serve as an accurate estimate of the lower bound for SQNR. We further verified the models with other parameter settings (N_c, L and N_f), and found similar correspondence between simulation and theory.

Given that the bit-accurate simulations match the theoretical results, we could use the methodology described in Section 5.1 to search for non-uniform bits allocation to optimise performance for a given resource budget.

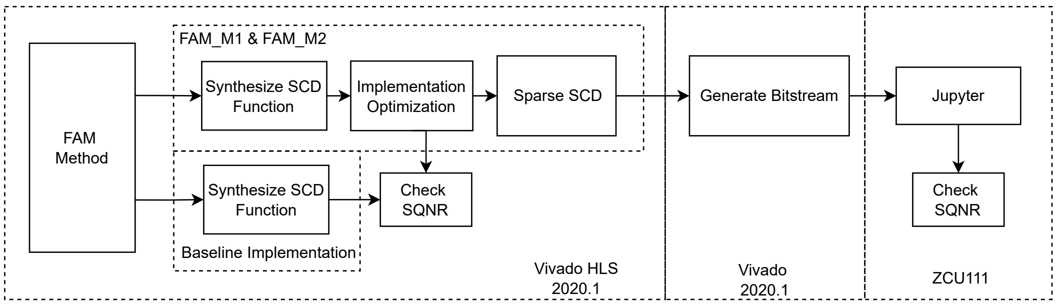


Fig. 15. Verification flow

The Vivado_HLS synthesis reports the LUTs and DSPs utilization for each design. The DSP48E2 slice contains a 27-bit by 18-bit two's complement multiplier. In Fig 14a, the DSPs utilization of FAM_M1 doubles when the wordlength is increased from 19 to 20 bits. This is because in our number system, 1 integer bit is used for the sign and the remaining 18 bits are used as fraction bits. For FAM_M2, as we assume that there are enough integer bits for the FFT sections and conjugate multiplication, the DSP utilization increases as the number of bits increases. In Fig 14a, at the same wordlength, FAM_M2 requires more DSPs than the FAM_M1. Hence in Fig 14b, we can see that at the same SQNR, FAM_M2 consumes fewer DSPs. Furthermore, based on our area model, we can look for a non-uniform allocation to balance DSP usage with the SQNR.

5.3 FPGA Implementation

We evaluate the performance of our design by comparing it with the state-of-the-art hybrid FPGA-GPU implementation [2] and the state-of-the-art GPU implementation [14], both of which perform alpha-profile calculations in terms of resource utilization, throughput, and power consumption. The alpha profile (Fig. 1c) is a one dimensional output but our sparse SCD (Fig. 1d), includes the frequency axis and, as explained in Section 4.3, provides richer information.

The verification process is shown in Fig. 15. Based on the parameter settings in the previous section, we first apply the baseline implementation of FAM method, then derive the FAM_M1 and FAM_M2 methods with the implementation optimization mentioned in Section 4.2 and Section 4.3 in Vivado HLS. Then we check the SQNR of the full DEMODULATE+FFT2 using a bit-accurate C simulation. We then generate a IP block for each of the two FAM methods with sparse SCD. Bitstreams are then generated in Vivado, tested via a Jupyter notebook which controls the execution of our FAM accelerators on the FPGA board. The output of the Jupyter notebook is compared with a floating point. We choose 16 and 24 bits as the wordlength for our design to trigger doubling of the number of DSPs required for implementation (see Fig. 14a). In terms of implementation, the data transfer between IP blocks will use AXI, which supports wordlengths in 16-bit multiples.

Table 5 gives a comparison of FPGA resource usage and operating frequency between a FPGA-GPU hybrid design [2], a recent FPGA Verilog design by Li et al. [13], the baseline implementation, and our optimized implementation. It can be seen that the hybrid FPGA-GPU design uses very few FPGA resources, since the FPGA is responsible for only a small part of the overall algorithm and it runs at a much lower clock frequency. While Li et al. [13] achieved a high clock frequency, it has higher resource utilization for the same wordlength. Even though our baseline implementation achieves a maximum clock frequency of 330 MHz, its II is 29 M cycles, so it takes 87.88 ms to process a window. In contrast, our optimized design achieves only a clock frequency of 200 MHz, but it takes only 33k clock cycles and has an execution time of 0.165 ms per window.

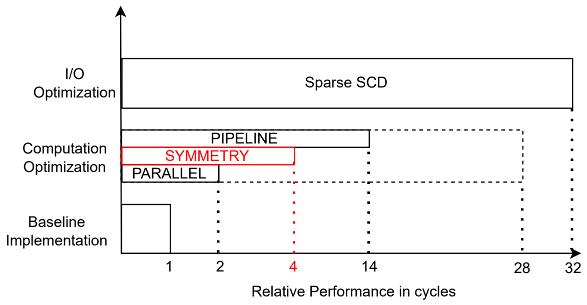


Fig. 16. Speedup breakdown for full-size SCD compared to baseline implementation in cycles. The total speedup is the product of all these optimizations.

Table 5. Comparison of FPGA resource usage and operating frequency

		wordlength	LUTs	FFs	BRAMs	DSPs	Power (W)	SQNR (dB)	Fmax
Full SCD	Hybrid FPGA-GPU design [2]	-	69 (0.1%)	153 (0.1%)	4 (2.9%)	0 (0%)	-	-	140MHz
	Avaliable on ZedBoard	-	53,200	106,400	140	220	-	-	-
Full SCD	Baseline implementation	16 bit	38,345 (9.0%)	46,557 (5.5%)	23 (2.1%)	226 (5.3%)	2.91[2]	71.05(70.88)	330MHz
	Optimized FAM_M1	16 bit	72,140 (17.0%)	63,699 (7.5%)	161 (14.9%)	736 (17.2%)	6.00[2]	5.41(3.83)[1]	200MHz
	Optimized FAM_M2	16 bit	85,050 (20.0%)	74,722 (8.8%)	162 (15.0%)	934 (21.9%)	6.10[2]	71.05(70.88)[1]	200MHz
Quarter SCD	Li et al. [13]	16 bit	150,802 (35.5%)	150,824 (17.7%)	264 (24.4%)	1,054 (24.7%)	12.5[2]	-	530MHz
	Optimized FAM_M2	16 bit	97,603 (23.0%)	89,477 (10.5%)	177 (16.4%)	1,048 (24.5%)	7.73[2]	71.05(70.88)[1]	200MHz
	Resources on ZCU111	-	425,280	850,560	1,080	4,272	-	-	-

[1] The theory values are in parentheses
[2] Total on-chip power estimated by Vivado

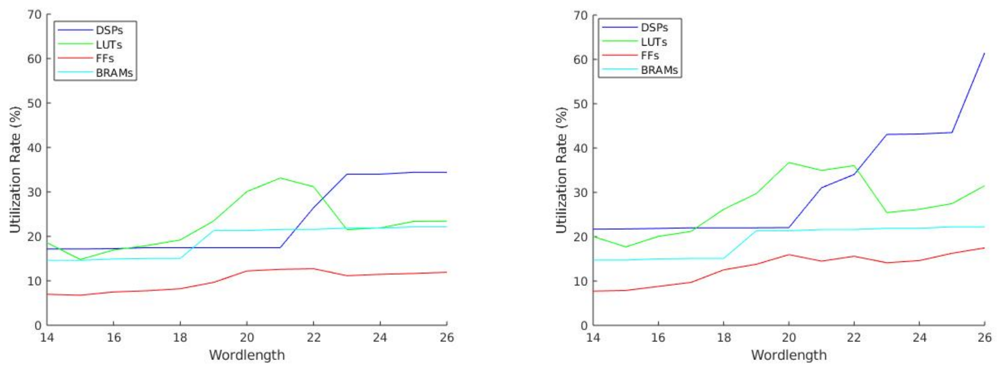
The optimization steps described in Section 4 include pipelining, parallelism, I/O and symmetry. Fig. 16 shows a bar chart with the performance gain achieved by each optimization. Since the computational bottleneck is FFT2, we set FSTRIDE to 2, doubling its performance. Pipelining minimizes the II and achieves a speedup of 14. The I/O optimization parallelises this stage with a speedup of 32 (value of P). Therefore the total speedup for the Full size SCD is the product of all of these factors, $2 \times 14 \times 32 = 896$. Finally, symmetry as shown in Fig. 16 allows us to avoid computing the full SCD, reducing computing for the Quarter SCD by a factor of 4, and improving the speedup to 3,584.

The FAM_M2 is also compared with FAM_M1. In Table 5, FAM_M2 achieves a higher SQNR for the same wordlength, but requires more resources. Thus, in the Fig. 17 the implementation resource utilization is plotted from 14 bits to 26 bits, showing that even at 24 bits FAM_M1 utilises more resource than FAM_M2 at 16 bis but still has a lower SQNR (53.32 dB in Fig 12), this indicates that the improvement in SQNR for FAM_M2 outweighs the additional resources. We also explored 16 bit, half precision floating point which uses 5 exponent bits and 10 fractional bits. This achieved an SQNR of 60 dB with 4× higher DSP utilization than the optimized 16 bit FAM_M2.

Referring to the FFT2 block in Fig. 5, we denote the number of real-valued multiply-accumulate (MAC) operations required per window for the conjugate multiplication, norm2, FFT2 and norm3 blocks as O_{CM}, O_{n2}, O_{FFT2} and O_{n3} respectively. Since these components account for the majority of FAM operations, we estimate the total number of MACs (N_{MAC}) as (Article 6 of reference [7]):

$$N_{MAC} \approx O_{CM} + O_{n2} + O_{FFT2} + O_{n3} \tag{34}$$

$$\approx 4N_c^2 N_f + 2N_c^2 N_f + 2N_c^2 N_f \log_2 N_f + 2N_c^2 N_f. \tag{35}$$



(a) FAM_M1 utilization rate

(b) FAM_M2 utilization rate

Fig. 17. Comparison of FPGA resource utilization between FAM_M1 and FAM_M2 (wordlengths from 14 bits to 26 bits)

Table 6. Comparison of throughput and power consumption for the same configuration of FAM.

	Full SCD[1]			Full SCD[2]	Quarter SCD[1]		Quarter SCD[2]
	GPU [2]	GPU [2]	FPGA+GPU [2]	Optimized	GPU [14]	FPGA [13]	Optimized
Platform	Tegra K1	Tesla K20	ZedBoard+Tegra K1	ZCU111	Tesla K40	ZCU111	ZCU111
Initiation Interval (ms)	111.61	8.98	50.95	**0.164**	0.303	0.065	**0.041**
Throughput (MS/s)	0.018	0.228	0.040	**12.5**	6.8	31.5	**50**
Speedup	1	12.3	2.1	**677.6**	366.7	1,704.4	**2,710**
Computational Performance (GOPS)	0.14	1.75	0.30	**460**	13.0	60.4	**460**
Power (W)	3.5	51	5	**35(6.10)**[5]	55.5[3]	12.5[4]	**37(7.65)**[5]
Energy Efficiency (MOPS/W)	40	34	60	**13,143**	234	4,832	**12,432**
Signal-to-quantization noise ratio (dB)	-	-	-	**73**[6]	-	-	**73**[6]

[1] Output is the alpha profile.
[2] Output is the sparse SCD.
[3] Power consumption is estimated by scaling to the result of [2].
[4] Power consumption is calculated from Vivado [13].
[5] The system power of entire ZCU111 board (power reported by Vivado report_power).
[6] An example of FAM_M2 using 16 bits. The system supports quantization error analysis for custom wordlengths.

Counting a MAC as 2 operations, number of cycles, $N_{cycles} = N_c^2$, and clock frequency $f_{clk} = 200\ MHz$, we estimate the numerical performance in billion operations per second to be

$$GOPS = \frac{2N_{MAC} \times f_{clk}}{N_{cycles}} \tag{36}$$

$$\approx 460. \tag{37}$$

When computing one quarter of the SCD, the $GOPS$ do not change as N_{cycles} and N_{MAC} are both reduced by the same amount. Furthermore, the designs in references [2, 13, 14] referenced in Table 6 calculate the alpha profile by first computing a full-sized or quarter SCD matrix.

We also measured the CPU performance of our FAM_M2 design using single precision floating point arithmetic. The GNU gcc compiler was used with "-Wall −std=c++14 -O3" parameters which gave best performance. On an Intel Core i7-9700 operating at 3.00GHz with 8 cores; Memory: 32 GB; and System: Ubuntu 18.04.5 LTS, a profile confirmed that 93% of the time was spent in FFT2 (15 ms).

Power consumption was measured using an Ecoflow River Pro inverter. In Table 6 we report the system power (power consumption measured at the AC power supply for the ZCU111 transformer) and total FPGA power (dynamic + static) as reported by the Vivado report_power command in

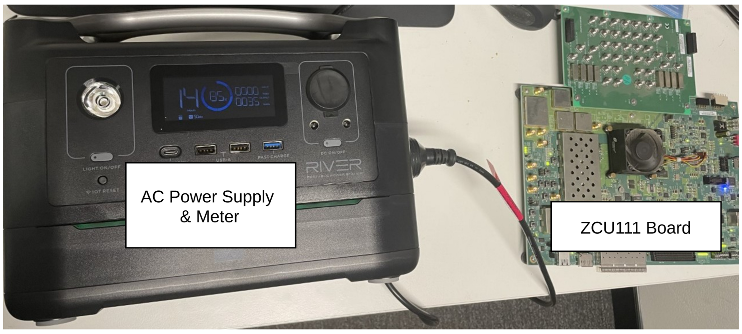


Fig. 18. Measuring the power consumption of the ZCU111 via AC power supply.

Table 7. Performance of two FAM methods running on FPGAs in 16 bits for different size of SCD matrices

	Size of SCD	SCD Function Time (ms)	Interface Delay (ms)	Execution Time (ms)	SQNR (dB)	Accuracy[1]
FAM_M2	Full	0.164	0.161	0.266	73.41	$2^{-13.7}$
	Quarter	0.041	0.055	0.088	73.66	2^{-14}

[1] The max error compared with floating point.

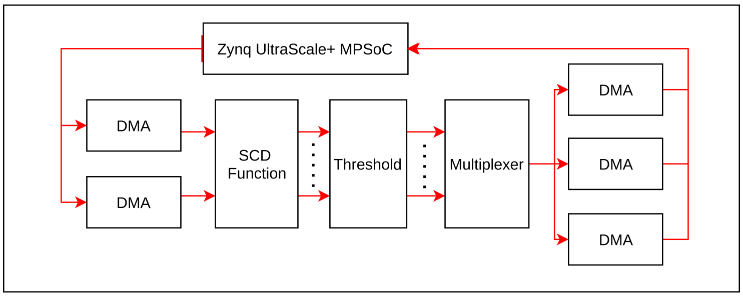


Fig. 19. Interface Delay.

parentheses. Fig. 18 shows the EcoFlow AC power supply powering the ZCU111. It is unclear to the authors whether the power consumption figures in reference [2] refer to system power or dynamic power, and power consumption is not reported in [14].

Li et al. [13] extrapolated Nvidia K40 performance (2,880 CUDA cores, 745 MHz clock) to a more recent RTX 3080 Ti GPU (8,960 CUDA cores, 1,365 MHz clock) and estimated a potential performance improvement of 5.5x [13]. Our HLS design can achieve this performance. However, the RTX 3080 Ti has a maximum power consumption of 350W so we believe our design will be more energy efficient.

Table 7 lists the execution time measured on the ZCU111 FPGA board. The execution time is the average time over 100 windows of input data, and larger than the FAM time due to data transfer overheads. Fig. 19 is a block diagram illustrating the interface delay of the system. To estimate the SCD function execution time, the threshold and multiplexer blocks were removed, keeping only the data transfer. In Table 7, it can be seen that the total execution time is close to the sum of SCD time and data transfer delay. The accuracy of the sparse SCD output is also calculated, and the maximum error compared to floating-point results for 16 bit data (15 fraction bits) were $2^{-13.7}$ and

2^{-14} for full-size or quarter-size sparse SCDs. We also verified that the sparse SCD matrix was the expected value.

6 CONCLUSION

In this paper, we derive explicit expressions to estimate signal to quantization noise, for the FAM technique in fixed precision (FAM_M1) and mixed precision (FAM_M2). This enables an understanding of how different blocks contribute to overall SQNR, and area-precision tradeoffs to be navigated in an analytical manner. Based on the quantization error analysis we note that FAM_M2 significantly improves SQNR with a minor increase in DSP resources (16-bit wordlength) and is hence preferable. Our simulations confirm that our analytic result matches the bit-exact simulation to within 1 dB. We also presented an HLS-based design of the FAM_M1 and FAM_M2 quantization schemes. High performance was achieved by exploiting spatial parallelism, pipelining, I/O optimization and symmetry. Together, these techniques allowed a design with state-of-the-art throughput and energy consumption.

In the future, we intend to extend the analysis to block floating point and integrate the IP core with data acquisition and deep neural networks for prediction.

ACKNOWLEDGMENTS

We thank the Xilinx University Program for their donation of a Zynq UltraScale+ RFSoC ZCU111 Evaluation Kit.

REFERENCES

[1] Jérôme Antoni, Ge Xin, and Nacer Hamzaoui. 2017. Fast computation of the spectral correlation. *Mechanical Systems and Signal Processing* 92 (2017), 248–277. https://doi.org/10.1016/j.ymssp.2017.01.011

[2] Nilangshu Bidyanta, G Vanhoy, M Hirzallah, A Akoglu, B Ryu, and T Bose. 2015. GPU and FPGA based architecture design for real-time signal classification. In *Proceedings of the 2015 Wireless Innovation Forum Conference on Wireless Communications Technologies and Software Defined Radio (WInnComm'15)*. Springer, San Diego, CA, 70–79.

[3] Ronald N. Bracewell. 1986. *The Fourier Transform and its applications.* McGraw Hill, Boston.

[4] Anders Brandt. 2011. *Noise and vibration analysis: signal analysis and experimental procedures.* John Wiley & Sons, Chichester, West Sussex, U.K. ;.

[5] William A Brown and Herschel H Loomis. 1993. Digital implementations of spectral correlation analyzers. *IEEE Transactions on Signal Processing* 41, 2 (1993), 703–720.

[6] William A Gardner. 1986. The spectral correlation theory of cyclostationary time-series. *Signal processing* 11, 1 (1986), 13–36.

[7] William A Gardner. 1994. *Cyclostationarity in communications and signal processing.* IEEE Press, New York.

[8] William A Gardner, Antonio Napolitano, and Luigi Paura. 2006. Cyclostationarity: Half a century of research. *Signal processing* 86, 4 (2006), 639–697.

[9] Pankaj Gupta. 2016. Accurate performance analysis of a fixed point FFT. In *2016 Twenty Second National Conference on Communication (NCC)*. IEEE, India, 1–6.

[10] DeepSig Inc. 2018. RF Datasets For Machine Learning. https://www.deepsig.ai/datasets.

[11] Xilinx Inc. 2019. Vivado Design Suite User Guide High-Level Synthesis UG902 (v2019.2). https://www.xilinx.com/support/documentation/sw_manuals/xilinx2019_2/ug902-vivado-high-level-synthesis.pdf

[12] Xilinx Inc. 2021. Fast Fourier Transform LogiCORE IP Product Guide PG109. https://docs.xilinx.com/v/u/en-US/pg109-xfft

[13] Xiangwei Li, Douglas L. Maskell, Carol Jingyi Li, Philip H.W. Leong, and David Boland. 2022. A Scalable Systolic Accelerator for Estimation of the Spectral Correlation Density Function and Its FPGA Implementation. *ACM Trans. Reconfigurable Technol. Syst.* (jun 2022). https://doi.org/10.1145/3546181 Just Accepted.

[14] Scott Marshall, Garrett Vanhoy, Ali Akoglu, Tamal Bose, and Bo Ryu. 2020. GPGPU based parallel implementation of spectral correlation density function. *Journal of Signal Processing Systems* 92, 1 (2020), 71–93.

[15] Razvan Nane, Vlad-Mihai Sima, Christian Pilato, Jongsok Choi, Blair Fort, Andrew Canis, Yu Ting Chen, Hsuan Hsiao, Stephen Brown, Fabrizio Ferrandi, et al. 2015. A survey and evaluation of FPGA high-level synthesis tools. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 35, 10 (2015), 1591–1604.

[16] Alan V Oppenheim and Clifford J Weinstein. 1972. Effects of finite register length in digital filtering and the fast Fourier transform. *Proc. IEEE* 60, 8 (1972), 957–976.

[17] Randy S Roberts, William A Brown, and Herschel H Loomis. 1991. Computationally efficient algorithms for cyclic spectral analysis. *IEEE Signal Processing Magazine* 8, 2 (1991), 38–49.

[18] Wolfgang Schlecker, Christiane Beuschel, and Hans-Jörg Pfleiderer. 2007. Quantisation noise in fixed-point multiplications. *Electrical Engineering* 89, 4 (2007), 339–342.

[19] Steven R Schnur. 2009. *Identification and classification of OFDM based signals using preamble correlation and cyclostationary feature extraction.* Technical Report. Naval Postgraduate School, Monterey CA.

[20] Dorde C Simic and JR Simic. 1999. The strip spectral correlation algorithm for spectral correlation estimation of digitally modulated signals. In *4th International Conference on Telecommunications in Modern Satellite, Cable and Broadcasting Services. TELSIKS'99*, Vol. 1. IEEE, Nis, 277–280.

[21] Chad Spooner. 2022. *J. Antoni's Fast Spectral Correlation Estimator.* https://cyclostationary.blog/2021/10/13/j-antonis-fast-spectral-correlation-estimator/

[22] Sundaramurthy and Reddy. 1977. Some Results in Fixed-Point Fast Fourier Transform Error Analysis. *IEEE transactions on computers* C-26, 3 (1977), 305–308.

[23] Bernard Widrow and Istvá Kollár. 2008. Quantization noise: roundoff error in digital computation, signal processing, control, and communications.

A QUANTIZATION ERROR ANALYSIS EXPRESSION FOR SCD FUNCTION

This appendix derives the output noise and signal variance of the FAM method for the Fixed Precision (FAM_M1) and Mixed Precision (FAM_M2) methods. The definition of the symbols in the expression are listed in the front (glossary of symbols). The σ^2_{FAM} is the variance of the output noise of FAM algorithm and P_{FAM} is the variance (power) of the output signal. In this appendix, the expressions of the output noise variance are simplified to the format of Eq. (27).

A.1 FAM_M1 - Fixed Precision Model

In this section, σ^2_W, $\sigma^2_{F_1}$, σ^2_{CM} and $\sigma^2_{F_2}$ are the noise variance generated by the quantization in windowing, first FFT, conjugate multiplication and second FFT section respectively. Similarly, G_{F_1}, G_{CM} and G_{F_2} are the gains of each section, used to amplify the noise and signal passed through. Details of the calculation of those parameters are described in the Section 3.1.1 and Section 3.3.

$$
\begin{aligned}
\sigma^2_{FAM\_M1} &= [(\sigma^2_W G_{F_1} + \sigma^2_{F_1})G_{CM} + \sigma^2_{CM}]G_{F_2} + \sigma^2_{F_2} \\
&= ((2\sigma^2_{s.CM}(\sigma^2_W \frac{1}{N_c} + \sigma^2_{F1}) + (\sigma^2_W \frac{1}{N_c} + \sigma^2_{F1})^2) + \sigma^2_r)\frac{1}{N_f} + \sigma^2_{F2} \\
&= (\frac{2P_{i.s}}{N_c 1.59^2}(\frac{2^{-2F_W}}{6N_c} + \frac{2^{-2F_1}}{3}(2 - \frac{m_1 + 1.5}{N_c})) + (\frac{2^{-2F_W}}{6N_c} + \frac{2^{-2F_1}}{3}(2 - \frac{m_1 + 1.5}{N_c}))^2 \\
&+ \frac{2^{-2F_{CM}}}{3})\frac{1}{N_f} + \frac{2^{-2F_2}}{3}(2 - \frac{m_2 + 1.5}{N_f}) \\
&\approx \frac{2P_{i.s}}{6N_c^2 N_f 1.59^2}2^{-2F_W} + \frac{2P_{i.s}(A_1)}{3N_c N_f 1.59^2}2^{-2F_1} + \frac{1}{3N_f}2^{-2F_{CM}} + \frac{A_2}{3}2^{-2F_2} \\
&= W_W 2^{-2F_W} + W_{F_1}2^{-2F_1} + W_{CM}2^{-2F_{CM}} + W_{F_2}2^{-2F_2}
\end{aligned}
\tag{38}
$$

$$
P_{FAM\_M1} = (\frac{P_{i.s}}{N_c 1.59^2})^2 \frac{1}{N_f} \tag{39}
$$

A.2 FAM_M2 - Mixed Precision Model

Expressions for the output noise and signal variance for the FAM method is given below.

$$\begin{aligned}\sigma^2_{FAM\_M2} &= [((((\sigma^2_W G_{F_1} + \sigma^2_{F_1})G_{Norm_1} + \sigma^2_{q_1})G_{CM} + \sigma^2_{CM})G_{Norm_2} + \sigma^2_{q_2})G_{F_2} + \sigma^2_{F_2}]G_{Norm_3} + \sigma^2_{q_3} \\ &\approx \frac{N_c^2 N_f P_{i.s} q_1^4 q_2^2 q_3^2}{3 * 1.59^2} 2^{-2F_W} + \frac{2P_{i.s} N_c N_f q_1^2 q_2^2 q_3^2}{3 * 1.59^2}[q_1^2 B_1 + 0.5]2^{-2F_1} \\ &+ \frac{N_f q_3^2 (q_2^2 + 0.5)}{3} 2^{-2F_{CM}} + \frac{q_3^2 B_2 + 0.5}{3} 2^{-2F_2} \\ &= W_W 2^{-2F_W} + W_{F_1} 2^{-2F_1} + W_{CM} 2^{-2F_{CM}} + W_{F_2} 2^{-2F_2}\end{aligned} \tag{40}$$

$$P_{FAM\_M2} = (\frac{P_{i.s} N_c}{1.59^2} q_1^2)^2 q_2^2 q_3^2 N_f \tag{41}$$