AMD Versal Implementations of FAM and SSCA Estimators

Carol Jingyi Li*†, Ruilin Wu*, Philip H.W. Leong*
* Computer Engineering Lab, The University of Sydney, NSW, Australia
† Reconfigurable Computing Systems Lab , The Hong Kong University of Science and Technology, Hong Kong
{jingyi.li, ruilin.wu, philip.leong}@sydney.edu.au

***Abstract*—Cyclostationary analysis is widely used in signal processing, particularly in the analysis of human-made signals, and spectral correlation density (SCD) is often used to characterise cyclostationarity. Unfortunately, for real-time applications, even utilising the fast Fourier transform (FFT), the high computational complexity associated with estimating the SCD limits its applicability. In this work, we present optimised, high-speed field-programmable gate array (FPGA) implementations of two SCD estimation techniques. Specifically, we present an implementation of the FFT accumulation method (FAM) running entirely on the AMD Versal AI engine (AIE) array. We also introduce an efficient implementation of the strip spectral correlation analyser (SSCA) that can be used for window sizes up to 2^{20}. For both techniques, a generalised methodology is presented to parallelise the computation while respecting memory size and data bandwidth constraints. Compared to an NVIDIA GeForce RTX 3090 graphics processing unit (GPU) which uses a similar 7nm technology to our FPGA, for the same accuracy, our FAM/SSCA implementations achieve speedups of 4.43x/1.90x and a 30.5x/24.5x improvement in energy efficiency.**

***Index Terms*—AIE, FPGA, Cyclostationary, FAM, SSCA.**

I. 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 characterisation of modulation types; noise analysis of periodic time-variant linear systems; synchronisation problems; parameter and waveform estimation; channel identification and equalisation; signal detection and classification; autoregressive (AR) and autoregressive moving average (ARMA) modelling and prediction; and source separation [1]–[3]. Cyclostationary analysis often involves estimating the spectral correlation density (SCD), which is the idealised temporal cross-correlation between all pairs of narrowband spectral components.

Although the SCD reveals extensive information about cyclostationary processes, the high computational requirements of the method poses problems for real-time applications. Even though the fast Fourier transform (FFT) significantly improves computational efficiency in the two most widely used SCD estimators, namely the FFT accumulation method (FAM) and strip spectral correlation analyser (SSCA) techniques, practitioners still seek enhanced performance to detect signals buried deep within noise [4], [5]. Consequently, there has been significant interest in developing high-performance implementations of the SCD method to detect and classify cyclostationary signals using central processing units (CPUs), graphics processing units (GPUs), and field-programmable gate arrays (FPGAs) technologies.

Traditional implementations of SCD algorithms on CPUs and GPUs are constrained by the fixed architectures of these platforms and are often not energy efficient. In contrast, FPGAs offer a flexible architecture that can be customised for specific applications. Moreover, they provide the possibility of integrating cyclostationary analysis with a software-defined radio and/or other signal processing and machine learning functionality. They also enable custom data paths that enhance parallelism and improve computational speed.

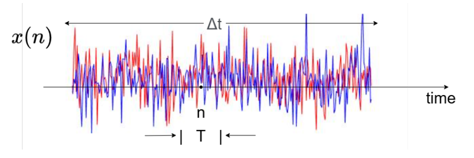
The AMD/Xilinx Versal ACAP architecture (Versal architecture), described in [6] merges general-purpose CPUs, programmable logic (PL), and AI Engine (AIE) processors optimised for AI and machine learning optimisation. With 400 AIE processors executing at a maximum 1.25 GHz, capable of delivering 8 MACs/cycles for 32-bit floating-point data, it has peak performance of 8 tera floating point operations per second (TFLOPS) [7]. The SCD estimators in this work were designed to later be integrated with an software-defined radio (SDR) front-end and machine learning (ML) back-end to perform radio frequency machine learning (RFML). The novel contributions of this paper are:

- A design methodology reported for high performance SCD estimation using the FAM and SSCA techniques on Versal platforms. The designs and results in this paper are reproducible[1].
- The first reported FAM implementation that only uses AIEs. Although performance is one-quarter of Ref. [8], it only requires 35% of the AIEs available and zero PL resources enabling future ML integration in RFML applications.
- An SSCA implementation employing a decomposed FFT and PL transpose unit to handle window sizes of the order of 1M samples. To the best of our knowledge, this is the first FPGA-accelerated SSCA implementation.

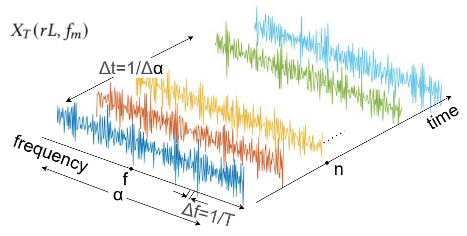
The remainder of this paper is structured as follows. Section II provides an overview of the SCD algorithms and the Versal architecture. In Section III, we detail the implementa-

The authors would like to thank the AMD/Xilinx University Program for the generous donation of a Versal VCK5000 board.

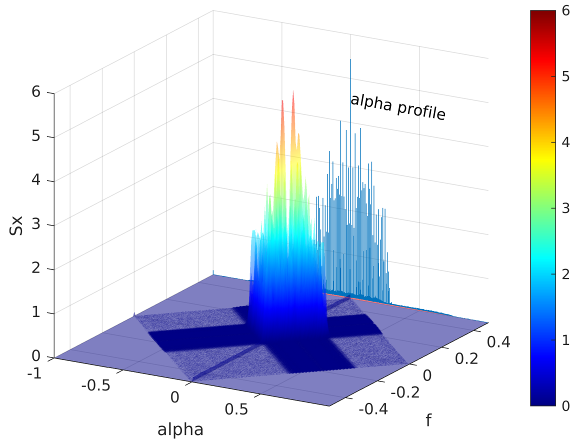
[1]The code can be found in https://github.com/Jingyi-li/SCD_VCK5000.



(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 $x(n)$ with alpha profile.

Fig. 1: The SCD function of direct-sequence spread-spectrum (DSSS) binary phase-shift keying (BPSK) signal.

tion based on AIE of two SCD algorithms on the AMD/Xilinx VCK5000 Versal platform (VCK5000). Section IV presents the experimental results, followed by the conclusions in Section V.

II. Background

A. Spectral Correlation Density

The description of the SCD function below follows that of Roberts et. al. [9] and Brown et. al [10]. 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

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

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.

B. FAM Technique

The direct application of Eq. (2) is computationally inefficient. Decimation and the FFT can be used to reduce the computational complexity [9]. Fig. 2 illustrates the signal flow for the FAM method, where the first task for both methods is

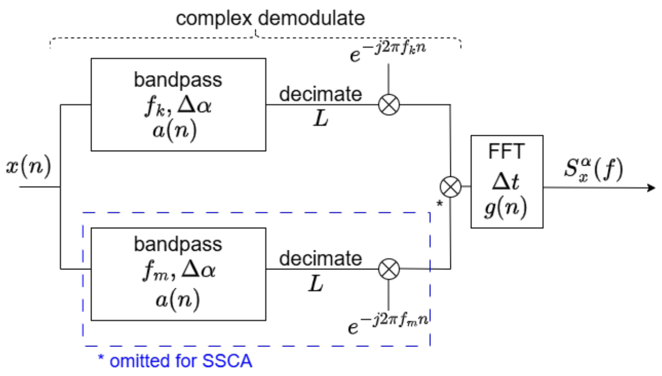


Fig. 2: Dataflow for FAM and SSCA (for SSCA $L = 1$ and the dashed block is omitted).

to compute the complex demodulates, X_T and Y_T (in Eq. (2)). We summarise the computations in this subsection, and refer readers to references [10], [11] for a detailed derivation, with implementation guidance in [12].

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 channeliser, with $L = N_P/4$ [10]. The Eq. (1) can be rewritten as

$$X_T(pL, f_m) = \underbrace{\Big[\sum_{k=0}^{N_P-1} \underbrace{a(d-k)x(pL-d+k)}_{x(n) \text{ windowed by } a(n)} e^{-i2\pi mk/N_P}\Big]}_{N_P \text{ point-FFT}} \underbrace{e^{-i2\pi mpL/N_P}}_{\text{Down Conversion}}, \tag{3}$$

where $p = \{0, 1..., P-1\}$, $d = N_P/2 - 1, r = d - k, f_m = mf_s/N_P$, $f_s = 1/T_s$, and $-N_P/2 < m < N_P/2$ [11]. Thus, in Eq. (3), the input is windowed via $a(n)$, then passed through a N_P-Point FFT. A phase shift is introduced to compensate for the down conversion from N to N_P samples.

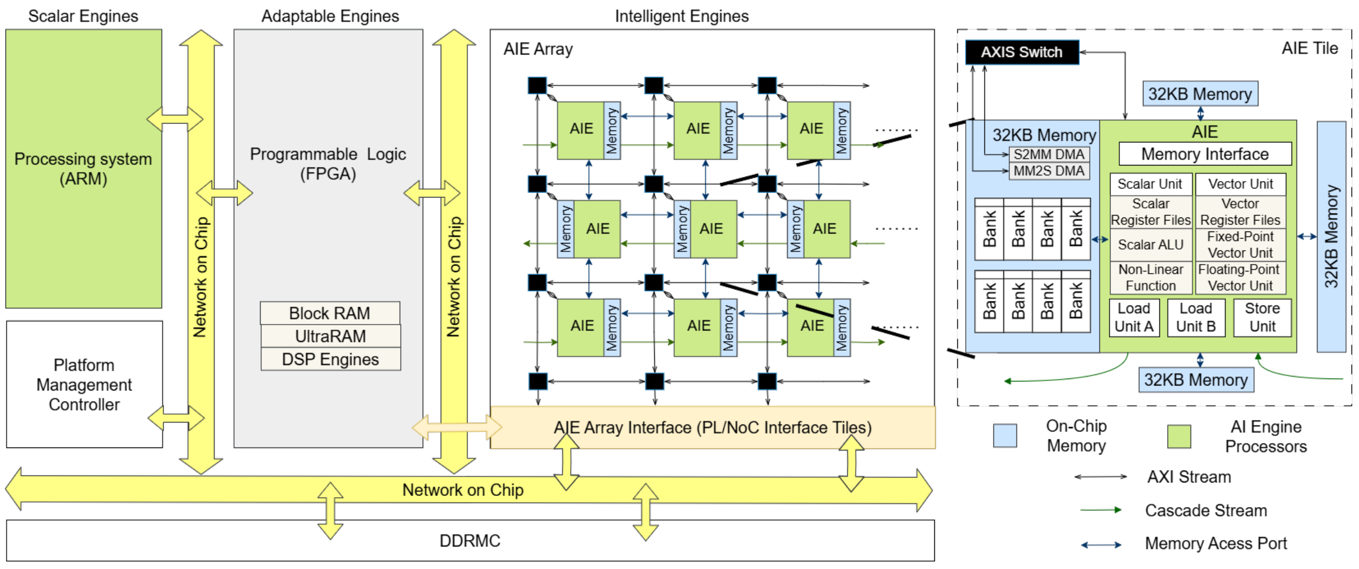


Fig. 3: AMD/Xilinx Versal ACAP Architecture.

2) FAM method: Taking Eq. (2), and substituting $X_T = Y_T$ to compute at the frequency $f_{kl} = (f_k + f_l)/2$ in P segments (Eq. (3)), Eq. (2) becomes

$$S^{\alpha_0}_{xy_T}(pL, f_{kl})_{\Delta t} = \sum_r X_T(rL, f_k)X^*_T(rL, f_l)g_d(p-r) \quad (4)$$

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. (4) and substituting $a_{kl} = f_k - f_l$, $f_0 = f_{kl} = (f_k + f_l)/2$, and $\alpha_0 = a_{kl} + q\Delta\alpha$ [11], the following is obtained

$$\begin{aligned} &S^{a_{kl}+\Delta\alpha}_x(pL, f_{kl})_{\Delta t} \\ &= \underbrace{\sum_r \underbrace{X_T(rL, f_k)X^*_T(rL, f_l)}_{\text{Conjugate Multiplication}} g_d(p-r)e^{-i2\pi rq/P}}_{\text{P-point FFT}}. \end{aligned} \quad (5)$$

C. SSCA Technique

Instead of multiplying two complex demodulates, the SSCA directly multiplies complex demodulates with the original signal [13]. In Fig. 2, the difference between SSCA and FAM is that instead of multiplying the complex demodulate $X_T(n, f_k)$ with $Y^*_T(n, f_k)$, it is multiplied by $y^*(n)$ to produce the channel-data product (CDP), X_g, for $k \in [-N_P/2, N_P/2-1]$. To ensure consistency in the sampling rate of the two terms, the channeliser decimation factor L is set to 1.

$$X_g(n+m, k) = X_T(n+m, f_k)x^*(n+m)g(m) \quad (6)$$

where $g(m)$ is a length $\Delta t = NT_s$ windowing function, and $m \in [-N/2, N/2-1]$. The centre frequencies of X_T are set to $f_k = k(f_s/N_P)$.

Finally, the N-point FFT of each of the N_P CDP values is computed resulting in the SCD estimate

$$S^{f_k+q\Delta\alpha}_X\left(\frac{f_k}{2} - q\frac{\Delta\alpha}{2}\right)_{\Delta t} = \underbrace{\sum_{m=-N/2}^{N/2-1} X_g(n+m, k)e^{-i2\pi qm/N}}_{\text{N-point FFT}} \quad (7)$$

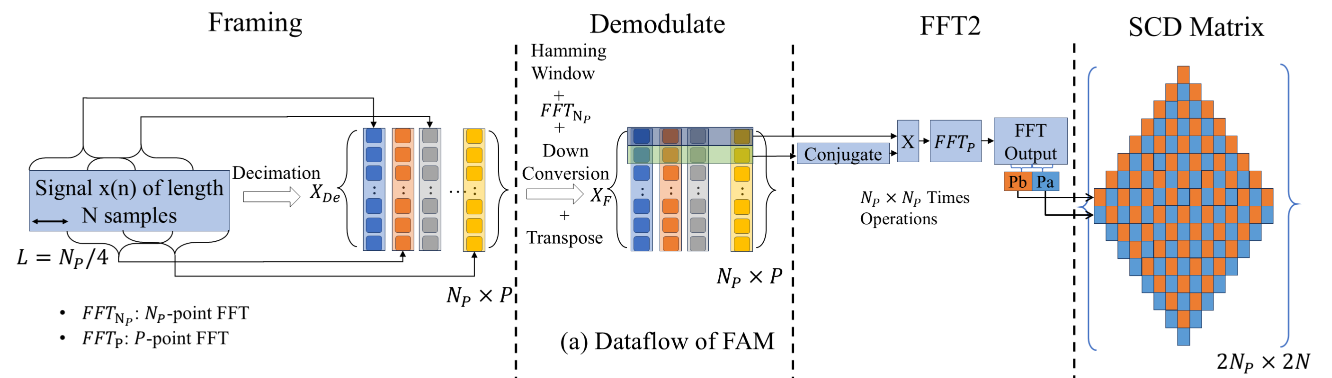
where cycle frequency $\alpha = f_k + q\Delta\alpha$, $\Delta\alpha = f_s/N$, $q \in [-N/2, N/2-1]$, and $f = (f_k - q\Delta\alpha)/2$ [11], [14]. In the implementation, both f and α are normalised based on $f_s = 1$, which maps $S^{\alpha}_X(f)$ to $f \in [-0.5, 0.5]$ and $\alpha \in [-1, 1]$.

D. Versal Architecture

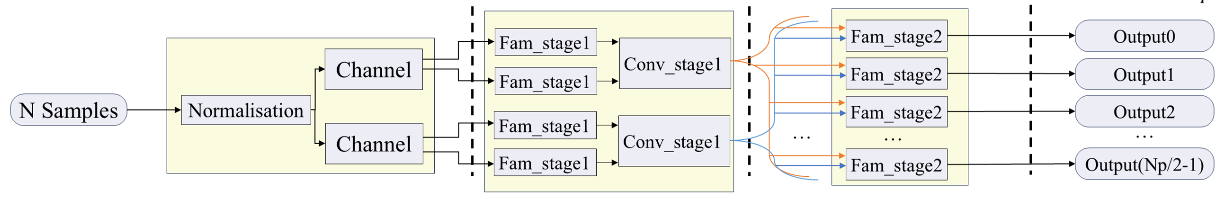
The AMD/Xilinx VCK5000 development card is powered by AMD's 7nm Versal™ Versal XCVC1902-2MSEVSVA2197 Adaptive SoC (VC1902), and AIE development is facilitated by the Vitis software platform [15].

Fig. 3 presents the overall Versal architecture, emphasising the AIE tile on the right [6]. The PL can be customised to meet specific application requirements, with digital signal processing (DSP) capabilities integrated for enhanced functionality. Additionally, the board incorporates an ARM processor for general-purpose processing tasks. The AIE array supports C/C++ programmability. The PL design can be made using RTL or C/C++ via high-level synthesis (HLS). [16]

These three components, the AIE array, ARM processor, and PL, are integral parts of the heterogeneous SoC. They operate independently and communicate through a network-on-chip (NoC) to other peripherals such as PCIe and DRAM controllers. The VCK5000 additionally features four DDR4 off-chip memory modules, each providing a peak bandwidth of 25.6 GB/s [6].


Framing
Demodulate
FFT2
SCD Matrix
Hamming Window + FFT_{N_P} + Down Conversion + Transpose
Signal x(n) of length N samples
$L = N_P/4$
Decimation
X_{De}
X_F
$N_P \times P$
Conjugate
X
FFT_P
FFT Output
Pb
Pa
$N_P \times N_P$ Times Operations
$2N_P \times 2N$
• FFT_{N_P}: N_P-point FFT
• FFT_P: P-point FFT

(a) Dataflow of FAM



(b) Parallelism Scheme for FAM on AIE

Fig. 4: Dataflow and parallelism scheme for FAM on VCK5000.

III. Method

The FAM method is limited by its nonuniform frequency resolution, making it suitable for analysing small-window signals where resolution is less critical [9], [17]. In contrast, the SSCA method provides uniform frequency resolution, making it appropriate for large-window signals analysis, with higher memory requirements [9], [10]. In this section, we present an AIE-based small-window FAM implementation optimised for processing speed (Sec. III-A), and a large-window SSCA implementation for applications that require accurate cycle frequency estimates (Sec. III-B).

A. *FAM Implementation*

Implementing the FAM entirely within AIE tiles allows for faster internal data transfers, avoiding a potential bottleneck between the PL and AIE. Thus, we focus on efficient utilisation of the limited AIE tile memory that is available. Key hardware constraints include:

- **AIE Tile Memory Constraints.** While an AIE tile can access data memory in four directions, a maximum of 2 input AXI4-Stream interfaces are supported, limiting the number of input buffers to two, i.e. the total input buffer capacity is $2 \times 32\,\text{KB} = 64\,\text{KB}$. Since each input buffer is organised in ping–pong A/B halves to overlap writes with computations, the actual size is halved again to 32 KB. Therefore, the usable size for each individual input buffer on an AIE Tile is 16 KB.
- **AIE array to PL Interface Constraints.** The VC1902 provides 39 columns to interface between the AIE array and PL, with each column having 6 streams (totaling 234 stream interfaces) [6].

Fig. 4 illustrates the FAM process from input samples to the final SCD matrix, covering three stages: **Framing**, **Demodulate**, and **FFT2**. Below, we analyse the relationships between parameters and optimisation details for the typical settings used in previous work [8], [12] ($N = 2,048$, $N_P = 256$) to facilitate an effective comparison.

1) Methodology: The small-window FAM architecture supports N_P values from 2^4 to 2^8, and input signal length N from 2^7 to 2^{12}. We define the total number of AIE tiles required to compute X_T in Eq. 3 as:

$$\mathbb{A}_{FAM} = \mathbb{A}_{Framing} + \mathbb{A}_{Demodulate} + \mathbb{A}_{FFT2}, \qquad (8)$$

where $\mathbb{A}_{Framing}$, $\mathbb{A}_{Demodulate}$ and $\mathbb{A}_{FFT2}$ are the number of tiles required for the Framing, Demodulate and FFT2 stages respectively. The intermediate matrix X_F between the demodulate stage and the FFT2 stage has a size of $N_P \times P$ complex values. Usually we set $L = N_P/4$ [10], so it follows that $P = N/L = 4N/N_P$. Therefore, the total size of the intermediate matrix X_F becomes $N_P P = 4N$ complex numbers. Each input buffer (up to 16 KB) stores $F = 2,048$ complex floats, needing $\lceil 4N/F \rceil$ buffers and `Fam_stage1` kernels in the Demodulate stage, plus $\lceil 4N/(2F) \rceil$ `Conv_stage1` kernels to aggregate data from the buffers, where $\lceil \cdot \rceil$ denotes the ceiling function.

The Framing stage employs one `norm` kernel for normalisation and $\lceil 4N/(2F) \rceil$ `Channel` kernels to distribute data to Demodulate kernels. The FFT2 stage involves conjugate multiplications and FFT computations. With a maximum of 234 AIE-to-PL streams, if $N_P > 234$ (e.g., 256), 128 kernels perform $N_P/128 \times N_P$ operations each, outputting results to 128 streams. Thus, the total tile count is:

$$\mathbb{A}_{FAM} = 1 + \left\lceil \frac{4N}{2F} \right\rceil + \left(\left\lceil \frac{4N}{2F} \right\rceil + \left\lceil \frac{4N}{F} \right\rceil \right) + \min(N_P, 128) \qquad (9)$$

Next, we will introduce the specific implementation of FAM on AIE array when $N = 2,048$, $N_P = 256$, and $P = 32$.

2) Framing Stage: Before decimation, the `norm` kernel normalises input $x(t)$. According to Eq. (3), the signal of length N is partitioned into P blocks, each containing N_P elements with offset $L = N_P/4$. The decimated matrix $X_{\text{De}} \in \mathbb{R}^{N_P \times P}$ is defined as:

$$X_{\text{De}}[n, p] = x[pL + n], \quad 0 \le n < N_P,\ 0 \le p < P.$$

The matrix X_{De} is column-wise partitioned into four equal submatrices $X_{\text{De}}^{(i)} \in \mathbb{C}^{N_P \times P/4}$ for parallel processing:

$$X_{\text{De}} = \left[X_{\text{De}}^{(0)} | X_{\text{De}}^{(1)} | X_{\text{De}}^{(2)} | X_{\text{De}}^{(3)}\right],$$

each mapped to independent kernels. The `channel` kernel routes data to Demodulate stage tiles.

3) Demodulate Stage: Based on Eq. (3), each column of the matrix from the Framing stage undergoes windowing, down conversion, and N_P-point FFT. Considering memory constraints (32KB per tile), this stage uses four `Fam_stage1` kernels, each processing $P/4$ frames, and two `Conv_stage1` kernels to broadcast data for FFT2.

Each kernel i processes frames in index set $\mathcal{J}_i$:

$$\mathcal{J}_i = \left\{ j \,|\, i\frac{P}{4} \le j < (i+1)\frac{P}{4} \right\}, \quad i = 0, 1, 2, 3.$$

Kernel outputs $X_T^{(i)}(f_m)$ correspond to these frame subsets:

$$X_T^{(i)}(f_m) = [\, X_T(jL, f_m)]_{j \in \mathcal{J}_i}, \quad i = 0, 1, 2, 3,$$

forming the combined output:

$$X_T(f_m) = [\, X_T^{(0)}(f_m) | X_T^{(1)}(f_m) | X_T^{(2)}(f_m) | X_T^{(3)}(f_m)\,].$$

In our design, the output from the Demodulate stage forms an $N_P \times P$ matrix, distributed across four kernels, each handling $P/4$ columns. Each `FAM_Stage1` kernel stores its column data into a buffer with a $P/4$-element stride. Thus, every $P/4$-element block in memory corresponds to $P/4$ points from one matrix row. Two `Conv` kernels then reorganise these blocks into four contiguous groups of $P/4$ points. The `FAM_Stage2` kernel receives the complete P-point input through two ports.

4) FFT2 Stage: For $N_P = 256$, 128 `FAM_stage2` kernels handle two frequency channels each:

$$f_{k_1} = 2i, \quad f_{k_2} = 2i + 1, \quad i = 0, \ldots, 127.$$

with the corresponding data $\mathbf{X}_T(rL, f_{k_1})$ and $\mathbf{X}_T(rL, f_{k_2})$ pre-loaded from `Conv_stage1` kernels. In FFT2 stage, the complex-conjugate data $\mathbf{X}_T^*(rL, f_l)$ is then broadcast *twice* to each kernel:

1) **Broadcast 1:** Multiply $\mathbf{X}_T(rL, f_{k_1})$ with $\mathbf{X}_T^*(rL, f_l)$.
2) **Broadcast 2:** Multiply $\mathbf{X}_T(rL, f_{k_2})$ with $\mathbf{X}_T^*(rL, f_l)$.

Mathematically, this expands the original single-pair operation $\{(k, l)\}$ into two passes for the pair $\{k_1, k_2\}$. For each `FAM_stage2` kernel, Eq. (5) becomes:

$$S_x^{(k_j, l)}(pL) = \sum_{r=0}^{P-1} X_T(rL, f_{k_j}) X_T^*(rL, f_l)\, g_d(p-r) e^{-i2\pi rq/P}.$$

By performing two broadcast passes of $\mathbf{X}_T^*(rL, f_l)$, each kernel can process the conjugate multiplications for its two assigned frequency channels $\{f_{k_1}, f_{k_2}\}$ without exceeding the available I/O resources.

Algorithm 1: AIE-based FAM pipeline pseudocode.

```
Function Framing(data_in):
    $data\_norm \leftarrow Normalize(data\_in)$;
    $X \leftarrow \text{zeros}(N_P, P)$;
    for $k = 0$ to $P - 1$ do
        $X(:, k+1) \leftarrow data\_norm[k \cdot L : k \cdot L + N_P - 1]$;
    return $X$;
Function Demodulate(X):
    $Y \leftarrow \text{zeros}(N_P, P)$;
    for $k = 0$ to $P - 1$ do
        $data\_win \leftarrow \text{Window}(chebwin[N_P], X(:, k))$;
        $data\_fft \leftarrow N_P\text{-Point FFT}(data\_win)$;
        $Y(:, k) \leftarrow \text{Down\_Conversion}(data\_fft, k)$;
    return $Y$;
Function FFT2(Y):
    for $m = 0$ to $N_P - 1$ do
        for $n = 0$ to $N_P - 1$ do
            $z \leftarrow Y(:, m) \cdot \text{conj}(Y(:, n))$;
            $z_{fft} \leftarrow P\text{-point FFT}(z)$;
            $z_{abs} \leftarrow |z_{fft}|^2$;
            write_to_output($z_{abs}[\frac{P}{2} .. \frac{3P}{4} - 1]$,
              $z_{abs}[\frac{P}{4} .. \frac{P}{2} - 1]$);
```

5) Implementation: We implement the FAM algorithm on VCK5000 platform. PL fetches 2,048 complex samples (64-bit, 8 KB) from DDR and streams them as two 32-bit AXI streams to AIE tiles. The entire algorithm is presented in Alg. 1.

B. SSCA Implementation

For very large SSCA windows (e.g., $N = 2^{20}$), intermediate matrices X_g must be stored on- or off-chip, and the AIE array primarily addresses computational complexity and store temporary variables. The key challenge on the VCK5000 is implementing an N-point FFT while managing memory bandwidth constraints.

1) SSCA_2DFFT: The strip spectral correlation analyser utilising a decomposed FFT (SSCA_2DFFT) is a novel approach that implements SSCA with a decomposed FFT (2DFFT), aligning the CDP computation order with the input sequence expected by the 2DFFT. This decomposition enables a more efficient mapping onto the AIE array, reducing intermediate matrix size and conserving memory bandwidth.

I. J. Good provided [18] the theoretical foundation for computing the Fourier transform of one-dimensional signals via multi-dimensional Fourier transforms [19]. The N-point discrete Fourier transform (DFT) can be computed using an $M_1 M_2$-point matrix DFT as:

$$\hat{x}(m_1', m_2') = \underbrace{\sum_{m_2=0}^{M_2-1} \underbrace{\{ \sum_{m_1=0}^{M_1-1} x(m_1, m_2) e^{-j2\pi \frac{m_1 m_1'}{M_1}} \}}_{M_1-\text{point DFT on } m_2\text{th column}} \underbrace{e^{-j2\pi \frac{m_2 m_1'}{M_2 M_1}}}_{\text{rotate factor}} e^{-j2\pi \frac{m_2 m_2'}{M_2}}}_{M_2-\text{point DFT on } m_1\text{th row}}, \tag{10}$$

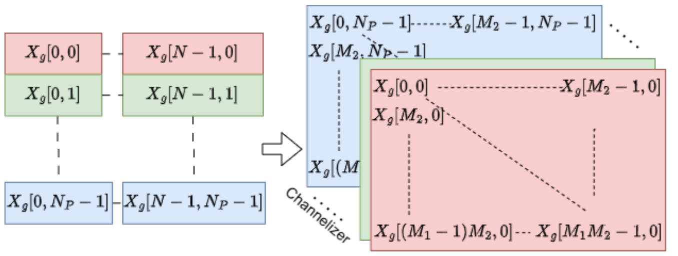


Fig. 5: Reshaping X_g from $[N \times N_P]$ to $[M_1 \times M_2 \times N_P]$, where $N = M_1M_2$.

where $M_1M_2 = N$, $0 \le m_1, m_1' < M_1$, and $0 \le m_2, m_2' < M_2$.

Fig. 5 shows how X_g is reshaped in SSCA_2DFFT to serve as input for the decomposed FFT stages, by mapping the 1D index to 2D coordinates for data reorganisation:

$$X_g^{2D}(m_1, m_2, k) = X_g(m_1M_2 + m_2, k), \tag{11}$$

where $0 \le m_1 < M_1$, $0 \le m_2 < M_2$ and $0 \le k < N_P$. The N_P channelizer computations run in parallel within X_g and can overlap with the 2DFFT, so the N_P 2DFFTs can be treated collectively, generating X_g only when needed.

The SSCA_2DFFT operates in two stages. **Stage 1** performs M_1-point FFTs on each of the M_1 rows of X_g, across all M_2 columns for N_P channelizers. During this stage, X_g is provided as required, and the outputs are then multiplied by a rotation factor and stored. **Stage 2** uses the results from Stage 1 to perform the M_2-point FFT for each row and channelizer. To achieve this, Eq. (3) becomes:

$$\begin{aligned} X_T^{2D}(m_1, m_2, f) &= X_T(m_1M_2 + m_2, f) \\ &= \sum_{r=-N_P/2}^{N_P/2-1} a(r)x(\eta + r)e^{-j2\pi f r T_s} * e^{-j2\pi f \eta T_s} \end{aligned} \tag{12}$$

where η is replaced by $\eta = m_1M_2 + m_2$. If M_2 is divisible by N_P, the down conversion term becomes $e^{-i2\pi f m_2 T_s}$ for all m_1. After multiplying by $x^*(\eta)$, the CDP is expressed as:

$$X_g^{2D}(m_1, m_2, k) = X_T^{2D}(m_1, m_2, f_k)x^*(\eta + m)g(m). \tag{13}$$

The final SSCA_2DFFT is then obtained by computing the 2DFFT of the N_P CDP values:

$$S_x^{2D}(m_1', m_2', k)_{\Delta t} = \textbf{Stage 2}(\textbf{Stage 1}(X_g^{2D}(m_1, m_2, k))), \tag{14}$$

in which the **Stage 1** is

$$\begin{aligned} &S_{x.s1}^{2D}(m_1', m_2, k)_{\Delta t} \\ &= \underbrace{\sum_{m_1=0}^{M_1-1} X_g^{2D}(m_1, m_2, k)e^{-j2\pi\frac{m_1m_1'}{M_1}}}_{M_1\text{-point FFT}} \underbrace{e^{-j2\pi\frac{m_2m_1'}{M_2M_1}}}_{\text{rotate factor}} \end{aligned} \tag{15}$$

and the **Stage 2** is

$$S_x^{2D}(m_1', m_2', k)_{\Delta t} = \underbrace{\sum_{m_2=0}^{M_2-1} S_{x.s1}^{2D}(m_1', m_2, k)_{\Delta t}e^{-j2\pi\frac{m_2m_2'}{M_2}}}_{M_2\text{-point FFT}}. \tag{16}$$

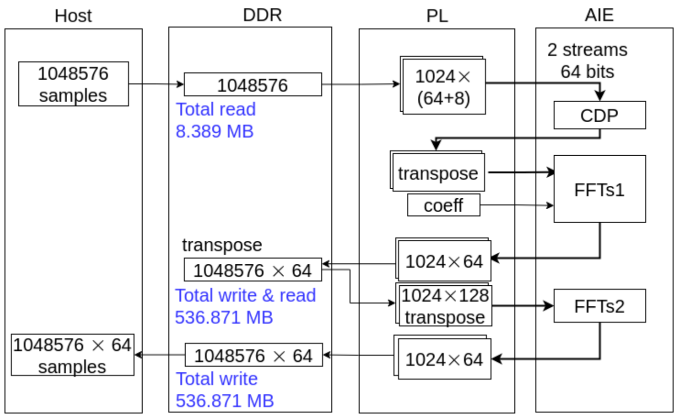


Fig. 6: Dataflow of SSCA_2DFFT on Versal.

Finally, $S_x^{2D}(m_1, m_2, k)$ is mapped to $S_x^{\alpha}(f)$ using:

$$f = \frac{k}{2N_P} - \frac{M_1m_2' + m_1' - N/2}{2N}$$

$$\alpha = \frac{k}{N_P} + \frac{M_1m_2' + m_1' - N/2}{N}.$$

2) Methodology: The VCK5000 platform offers 23.9 MB of on-chip SRAM and four 4 GB off-chip DDR. Since an intermediate matrix in single-precision complex format requires $8 \times N \times N_P$ bytes, off-chip memory is needed if $N \times N_P > 2^{20}$.

Each AIE tile features eight single-port memory banks (32KB total), sufficient to perform two-step 1K (1,024-element) single-precision complex computations using a ping-pong buffer scheme. In the SSCA implementation, the number of N_P channelizers ranges from 2^5 to 2^8, and the size of the second N-point FFT spans from 2^{12} to 2^{20}. Because each AIE tile is efficient at processing a 1K array, we assign $\mathbb{A}_{CDP}$ AIE tiles to compute $1K/N_P$ window sets of X_g (Eq. 6) in parallel. Accordingly, $\mathbb{A}_{CDP}$ is given by:

$$\mathbb{A}_{CDP} = 1 + \lceil \log_2(N_P)/2 \rceil, \tag{17}$$

with one for down conversion/conjugate multiplication, and the rest tiles for N_P-point FFT. The window function is merged into the first FFT stage.

For the large-point FFT, the N-point FFT is decomposed into M_1 and $M_2 = N/M_1$. Thus, the number of AIE tiles required is:

$$\mathbb{A}_{2DFFT} = \lceil \log_2(M_1)/2 \rceil + 1 + \lceil \log_2(M_2)/2 \rceil, \tag{18}$$

where $M_1,\ M_2 < 1K$. Additionally, one extra tile is allocated for multiplication with the rotation factors. The design computes $1K/M_1$ instances of M_1-point FFT and $1K/M_2$ instances of M_2-point FFT in parallel.

Thus, the total AIE tiles needed for SSCA_2DFFT is

$$\mathbb{A}_{SSCA} = \mathbb{A}_{CDP} + \mathbb{A}_{2DFFT}. \tag{19}$$

Replicating $\mathbb{A}_{SSCA}$ modules enables further parallelisation.

Algorithm 2: AIE-based SSCA_2DFFT pseudocode.

```
Function CDP ($datain\_even, datain\_odd$):
    $data\_win$, $data\_fft$, $data\_dc$, $data\_out$ =
      $zeros(M_1, 1)$;                                ▷ $M_1 = 16N_P$
    $xc = zeros(16, 1)$;
    static int $itr = 0$;
    $data\_win, xc \leftarrow$
      Window($chebwin[N_P], datain\_even, datain\_odd$);
    $data\_fft \leftarrow$ $\mathbf{N_P}$-dimensional FFT($data\_win$);
    $data\_dc \leftarrow$ Down_Conversion($data\_fft$, int($itr/N_P$));
    $data\_out = data\_dc \times xc$;
    $itr = (itr == (M_2 * N_P))?\,0 :\ (itr + 1)$;
    return $data\_out$;
Function FFTs1 ($datain\_even, datain\_odd, tow$):
    $data\_fft$, $data\_out$, $rotate\_factor = zeros(M_1, 1)$;
    if $itr\%N_P == 0$ then
        $rotate\_factor \leftarrow Compute\_rotate\_factor(tow)$;
    $data\_fft \leftarrow$
      $\mathbf{M_1}$-dimensional FFT($data\_even$, $data\_odd$);
    $data\_out = data\_fft \times rotate\_factor$;
    $itr = (itr == (M_2 * N_P))?\,0 :\ (itr + 1)$;
    return $data\_out$;
Function FFTs2 ($datain\_even, datain\_odd, tow$):
    $data\_out = zeros(M_2, 1)$;
    $data\_out \leftarrow$
      $\mathbf{M_2}$-dimensional FFT($data\_even$, $data\_odd$);
    return $data\_out$;
```

3) Implementation: We developed a complete SSCA implementation with $N = 1,048,576$ (i.e., an example of one million input size), $N_P = 64$, and $M_1 = M_2 = 1,024$ to balance the number of iterations between `FFTs1` and `FFTs2`. Fig. 6 shows the overall architecture on the VCK5000 platform. The PL supports data transfer from the DDR memory controller (DDRMC) to the AIE array, and between sections in AIE tiles. Our implementation of a single-precision, large N-point FFT uses ideas from Ref. [20] but: (1) loads the data from CDP, (2) matches the bandwidth of the DDRMC, and (3) minimises DDR access time by optimising row accesses. Previous works such as Ref. [21] used fixed-point arithmetic.

Alg. 2 outlines the SSCA_2DFFT implementation on the AIE array. The PL streams input to the `CDP` stage, which performs windowing, N_P-point FFTs, down conversion, and conjugate multiplication. The $M_1 \times N_P$ output matrix is transposed in PL and forwarded to `FFTs1`, where M_1-point FFTs and rotate factors are applied across M_2 iterations. Results exceeding on-chip memory capacity ($N_P \times M_1 \times M_2$) are stored in DDR. Then PL reads the data for `FFTs2` using a stride access pattern to achieve transpose input.

The PL manages data transfers between DDRMC and AIE array. To fully utilise the bandwidth, a 512-bit data bus transfers eight 64-bit complex samples per cycle. For `CDP`, the PL uses a ping-pong scheme using two algernative buffers B_0 and $B_1 \in \mathbb{C}^{M_1 \times (N_P + LANE)}$, where $LANE = 8$. While one buffer is filled by the PL, the other is read by the AIE, alternating each batch. The buffer accommodates $LANE \cdot N_P$ iterations for `CDP` per load, ensuring continuous data supply to the `CDP` module with minimal stalling.

To match the input format of `FFTs1`, the `CDP` output (an $[M_1 \times N_P]$ matrix) is transposed in the PL. A ping-pong buffer enables parallel loading and forwarding, hiding latency and maintaining continuous dataflow to `FFTs1`.

The PL also manages ping-pong buffers for storing `FFTs1` outputs and loading inputs to `FFTs2` via DDR. While storage to DDR is sequential, loading requires transposed access. To avoid inefficient strided reads, we allocated a larger buffer to fetch blocks of size $[M_2 \times CN_P]$ instead of $[M_2 \times N_P]$, improving access efficiency by a factor of C.

IV. Results

We implement the FAM and SSCA designs described in Sec. III on the VCK5000 platform, with AIE running at 1 GHz and PL running at 312.5 MHz. We then compare their performance against a conventional implementation on an Intel(R) Xeon(R) Silver 4208 CPU and NVIDIA GeForce RTX 3090, both at 2.10 GHz under Ubuntu 22.04.4 LTS.

A. Accuracy

Accuracy was tested using a DSSS BPSK signal with 10 dB signal-to-noise ratio (SNR), processing gain of 31, chip rate 0.25 and sample rate normalised to 1, resulting in cycle frequencies that are multiples of the data rate (0.25/31). We used IEEE 754 double-precision MATLAB results as a reference for validation. When comparing the MATLAB and VCK5000 implementations, the FAM algorithm achieves an average relative error of 9.94e-5. We compare the VCK5000-based SSCA_2DFFT with a CPU SSCA implementation written in C++ that used the same FFT coefficients. Under these conditions, the average relative error reduced to 1.08e-6.

B. Utilisation

Tab. I shows the utilisation of resources. The FAM design does not require URAM, as buffering is managed within AIE tiles. For the SSCA, BRAM and URAM are used, mainly for the ping-pong buffer between the DDRMC and AIE components.

In the FAM implementation, 3 AIE tiles are allocated for signal normalisation and decimation in the Framing stage. The Demodulate stage employs 6 AIE tiles to perform windowing, N_P-point FFT, and down conversion. In the final FFT2 stage, since $N_P > 128$, 128 AIE tiles are used to execute the conjugate multiplication and P-point FFT. As shown in Tab. I, the implementation of the FAM algorithm requires 137 AIE tiles, which is consistent with the value in Eq. (9). In this case, all computational kernels are executed on the AIE array, making the design easily portable to AIE-only platforms.

In the SSCA implementation, the `CDP` module requires 4 AIE tiles that compute M_1/N_P sets of N_P data in each iteration, and this utilises a ping-pong buffering scheme to exchange data between tiles. In the `FFTs1` and `FFTs2` modules, 5 AIE tiles are used to compute the M_1-point FFT, with an additional tile in `FFTs1` for the computation of rotation factors. This totals 15 AIE tiles (Tab. I), matching Eq. (19).

TABLE I: Utilisation in VCK5000

	PL					AIE Array	
	Register	LUT	LUT as MEM	BRAM	URAM	AIE tile	PLIO
Total resources	1,739,432	860,336	446,367	933	463	400	-
FAM	113,686 (6.61%)	107,601 (12.73%)	960 (0.22%)	37 (3.97%)	0 (0.00%)	137 (34.25%)	130
SSCA_2DFFT	15,475 (0.89%)	11,824 (1.37%)	1,575 (0.35%)	349 (37.41%)	192 (41.47%)	15 (3.75%)	13

TABLE II: Comparison with other FAM implementations

	[12]	[8]	Our
Platform	ZCU111	ZCU111	VCK5000
Initiation Interval (ms)	0.26	0.164	0.63
Throughput (MS/s)	7.88	12.50	3.25
Computational Performance (GOPS)	60.40	460	189
Board Power (W)	12.50[1]	35	40

[1] Chip rather than board power.

TABLE III: Execution time and speedup vs CPU and GPU

	FAM		SSCA	
	Time	Speedup	Time	Speedup
CPU	0.194 s	1	11.3 s	1
GPU	2.791 ms	69.51	217 ms	52.07
VCK5000	0.630 ms	307.94	114 ms	99.12

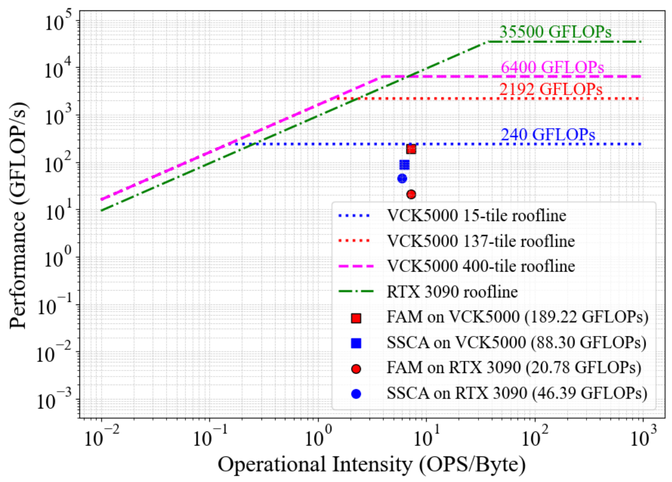


Fig. 7: Rooflines of our Implementations.

C. Performance

The code running on the CPU was compiled using g++ version 9.4.0 with the "-O2" optimisation flag. The GPU code was compiled using nvcc, with the host compiler set to g++. The compilation targeted CUDA architecture "sm_86", using C++17 standard with the "-O3" optimisation flag. Additionally, the code is linked with the cuFFT library to support efficient FFT operations [22].

In Tab. II we compare our implementation with existing FPGA designs. In Ref. [12], a quarter SCD is implemented for comparison with our implementation. To enable a fair comparison with our full SCD, the reported initiation interval was scaled by a factor of four.

Tab. III presents the execution times (measured on the physical platform) for the SSCA and FAM algorithms on CPU, GPU, and VCK5000 platforms. For FAM, the VCK5000 showed a speedup of 308x compared to CPU and 4.43x over GPU. For SSCA, the VCK5000 achieved a speedup of 99.12x over the CPU and 1.90x over the GPU.

As shown in the roofline plot of Fig. 7, the FAM implementation on VCK5000 achieves 189 GFLOPs, corresponding to 8.6% of the 137-tile peak performance (2192 GFLOPs). The SSCA implementation reaches 88.30 GFLOPs, achieving 37% of its 15-tile peak (240 GFLOPs). The relatively low utilisation of the FAM design is attributed to its architectural design: although 137 AIE tiles are allocated, a significant portion of them are used solely for data movement rather than floating-point operations. This leads to the under-utilisation of available computational resources. The performance of SSCA is constrained by the bandwidth between PL and DDRMC. In our system, communication with 15 AIE tiles saturates this available bandwidth. With increased off-chip bandwidth, additional AIE tiles could be utilised to further enhance performance. On the RTX 3090, both FAM and SSCA implementations are memory-bound, achieving 20.78 GFLOPs and 46.39 GFLOPs respectively, also far below the GPU's ceiling of 35 TFLOPs.

We measure power consumption on the VCK5000 and GPU using the "xbutil" and "nvidia-smi" command-line tools, respectively. For FAM and SSCA, the VCK5000 consumes 17 W and 8 W, respectively, on top of an idle power of 23 W. This value is consistent with the power estimate from Xilinx Power Design Manager which reported 13.7 W for FAM. The GPU requires 117 W and 103 W with an idle of 33 W. Consequently, compared to GPU, the VCK5000 achieves a 30.5x higher energy efficiency for FAM and 24.5x higher for SSCA. Our SSCA design requires fewer AIE tiles, resulting in lower power usage on the VCK5000.

V. Conclusion

This paper presents a novel design methodology for high-speed implementations of the FAM and the SSCA on the Versal platform. For the SSCA implementation, we take advantage of the heterogeneous nature of the Versal architecture and utilise the AIE array's parallel compute capabilities in parallel with the PL to minimise data transfers and manage large intermediate matrices. Our design demonstrates the potential of Versal devices for real-time cyclostationary signal analysis, paving the way for future integration with SDR front-ends and machine learning back-ends in advanced RFML applications.

References

[1] W. A. Gardner, A. Napolitano, and L. Paura, “Cyclostationarity: Half a century of research,” *Signal processing*, vol. 86, no. 4, pp. 639–697, 2006.

[2] B. Ramkumar, “Automatic modulation classification for cognitive radios using cyclic feature detection,” *IEEE Circuits and Systems Magazine*, vol. 9, no. 2, pp. 27–45, 2009.

[3] X. Liu, C. J. Li, C. T. Jin, and P. H. W. Leong, “Wireless signal representation techniques for automatic modulation classification,” *IEEE Access*, vol. 10, pp. 84 166–84 187, 2022.

[4] W. A. Gardner, “Exploitation of spectral redundancy in cyclostationary signals,” *IEEE Signal Processing Magazine*, vol. 8, pp. 14–36, Apr. 1991.

[5] W. A. Gardner, “The spectral correlation theory of cyclostationary time-series,” *Signal processing*, vol. 11, no. 1, pp. 13–36, 1986.

[6] AMD Xilinx, *AM009 Versal AI Engine*, 2021, versal ACAP AI Engine Architecture Manual. [Online]. Available: https://www.xilinx.com

[7] ——, *XMP452 Versal AI Core Series Product Selection Guide*, 2024, versal AI Core Series Product Selection Guide. [Online]. Available: https://www.xilinx.com

[8] C. J. Li, X. Li, B. Lou, C. T. Jin, D. Boland, and P. H. W. Leong, “Fixed-point fpga implementation of the fft accumulation method for real-time cyclostationary analysis,” *ACM Trans. Reconfigurable Technol. Syst.*, vol. 16, no. 3, Jun. 2023. [Online]. Available: https://doi.org/10.1145/3567429

[9] R. S. Roberts, W. A. Brown, and H. H. Loomis, “Computationally efficient algorithms for cyclic spectral analysis,” *IEEE Signal Processing Magazine*, vol. 8, no. 2, pp. 38–49, 1991.

[10] W. A. Brown and H. H. Loomis, “Digital implementations of spectral correlation analyzers,” *IEEE Transactions on Signal Processing*, vol. 41, no. 2, pp. 703–720, 1993.

[11] W. A. Gardner, *Cyclostationarity in communications and signal processing*. New York: IEEE Press, 1994.

[12] X. Li, D. L. Maskell, C. J. Li, P. H. W. Leong, and D. Boland, “A scalable systolic accelerator for estimation of the spectral correlation density function and its fpga implementation,” *ACM Trans. Reconfigurable Technol. Syst.*, vol. 16, no. 1, Dec. 2022. [Online]. Available: https://doi.org/10.1145/3546181

[13] W. A. Brown, “On the theory of cyclostationary signals,” Ph.D. dissertation, University of California Davis, 1987.

[14] E. April, *On the Implementation of the Strip Spectral Correlation Algorithm for Cyclic Spectrum Estimation*, ser. DREO technical note. Defence Research Establishment Ottawa, 1994. [Online]. Available: https://books.google.com.au/books?id=7QD7MwEACAAJ

[15] “Vck5000 versal development card,” https://www.xilinx.com/products/boards-and-kits/vck5000.html, accessed: 2025-03-28.

[16] AMD Xilinx, *UG1076 Versal ACAP AI Engine Programming Environment User Guide*, 2022, running Software Emulation chapter. [Online]. Available: https://docs.amd.com/r/2022.1-English/ug1076-ai-engine-environment/Running-the-System-in-Hardware

[17] J. Antoni, G. Xin, and N. Hamzaoui, “Fast computation of the spectral correlation,” *Mechanical Systems and Signal Processing*, vol. 92, pp. 248–277, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0888327017300134

[18] I. J. Good, “The interaction algorithm and practical fourier analysis,” *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 20, no. 2, pp. 361–372, 12 2018. [Online]. Available: https://doi.org/10.1111/j.2517-6161.1958.tb00300.x

[19] L. R. Rabiner and B. Gold, *Theory and application of digital signal processing*. Englewood Cliffs, N.J: Prentice-Hall, 1975.

[20] AMD Xilinx, “1 million point float FFT @ 32 Gsps on AI Engine,” 2024, accessed: 2025-03-28. [Online]. Available: https://github.com/Xilinx/Vitis-Tutorials/tree/2024.2/AI_Engine_Development/AIE/Design_Tutorials/16-1M-Point-FFT-32Gsps

[21] H. Kanders, T. Mellqvist, M. Garrido, K. Palmkvist, and O. Gustafsson, “A 1 million-point FFT on a single fpga,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 66, no. 10, pp. 3863–3873, 2019.

[22] “cuFFT API reference,” https://docs.nvidia.com/cuda/cufft/, accessed: 2025-03-28.