Hardware–Software Co-Design of Statistical and Deep-Learning Frameworks for Wideband Sensing on Zynq System on Chip

Rohith Rajesh, Sumit J. Darak, Akshay Jain, Shivam Chandhok, and Animesh Sharma

Abstract—With the introduction of spectrum sharing and heterogeneous services in next-generation networks, the base stations need to sense the wideband spectrum and identify the spectrum resources to meet the quality-of-service, bandwidth, and latency constraints. Sub-Nyquist sampling (SNS) enables digitization for sparse wideband spectrum without needing Nyquist speed analog-to-digital converters (ADCs). However, SNS demands additional signal processing algorithms for spectrum reconstruction, such as the well-known orthogonal matching pursuit (OMP) algorithm. OMP is also widely used in other compressed sensing applications. The first contribution of this work is efficiently mapping the OMP algorithm on the Zynq system-on-chip (ZSoC) consisting of an ARM processor and field-programmable gate array (FPGA). Experimental analysis shows a significant degradation in OMP performance for sparse spectrum. Also, OMP needs prior knowledge of spectrum sparsity. We address these challenges via deep-learning (DL)-based architectures and efficiently map them on the ZSoC platform as a second contribution. Via hardware–software codesign (HSCD), different versions of the proposed architecture obtained by partitioning between software (SW) (ARM processor) and hardware (HW) (FPGA) are considered. The resource, power, and execution time comparisons for given memory constraints and a wide range of word lengths (WLs) are presented for these architectures.

Index Terms—Convolutional neural network (CNN), deep learning (DL), hardware–software codesign (HSCD), orthogonal matching pursuit (OMP), sub-Nyquist sampling (SNS), Zynq system-on-chip (ZSoC).

I. INTRODUCTION

WIDEBAND spectrum sensing (WSS) involves digitizing the wideband spectrum and identifying available spectrum for resource allocation in wireless networks [1], [2], [3]. WSS has gained significant importance in 5G and next-generation wireless networks due to the introduction of spectrum-sharing policies replacing the conventional static spectrum allocation policies [4]. Spectrum sharing allows the deployment of wireless networks in licensed, shared, and unlicensed spectrum [5]. The advantages include significant cost savings for Telecom service operators since they can reduce licensed spectrum requirements and corresponding exorbitant spectrum license fees, usually billions of dollars [6]. A broader spectrum enables the deployment of upcoming heterogeneous services that demand a wide range of bandwidth, quality-of-service, and latency constraints. Though the millimeter-wave spectrum (above 6 GHz) is being explored due to the availability of large bandwidth, various studies have confirmed that it is not a feasible alternative to the sub-6-GHz spectrum for reliable outdoor communication and comprehensive network coverage.

Numerous spectrum measurement and utilization studies have shown that overall utilization of the sub-6-GHz spectrum is poor even though most of the spectrum is licensed [7], [8], [9]. For digitization of such a sparse wideband spectrum, sub-Nyquist sampling (SNS)-based WSS is an efficient and feasible alternative to Nyquist sampling-based WSS. This is because SNS needs low-speed analog-to-digital converters (ADCs) compared to Nyquist rate ADCs in the latter [2], [10]. However, SNS-based WSS needs additional digital signal processing to recover the SNS sampled spectrum so that it closely resembles the original spectrum [11], [12]. Such recovery must be done accurately and should meet the stringent area, power, and latency constraints.

Various signal recovery techniques for SNS have been reviewed in [11]. The greedy-approach-based orthogonal matching pursuit (OMP) framework is popular due to its lower complexity and faster execution. Other applications of OMP include image processing and radar systems [13], [14], [15], [16]. Recently, advances in deep learning (DL) have been explored to improve the performance of the OMP [17], [18], [19]. Most of these approaches augment the OMP by replacing the matched filtering task with the DL. The OMP and its variants suffer from a significant degradation in reconstruction performance, especially when the spectrum sparsity is high. Also, they need prior knowledge of spectrum sparsity which may not be available in a dynamic environment.

In the ongoing and proposed deployments of 5G [20], [21], [22], [23], the base station has been restructured to support open radio access network (O-RAN), enable interoperability,
and low-latency high-throughput services. In this direction, base stations are now divided into three parts: radio unit (RU), distributed unit (DU), and centralized unit (CU). Depending on the deployment scenario such as rural, urban, and type of services (gaming, voice, and video), DU may or may not be colocated with CU or RU when compared to 4G where CU and DU were colocated to form a baseband unit (BBU). Thus, there is a need to realize RU and DU functionality on the system-on-chip (SoC) due to their remote locations and stringent constraints on their size, weight, and power consumption. Furthermore, RU and DU are responsible for real-time functions that demand efficient mapping on the SoC [24], [25], [26], [27].

The first contribution of this work is to efficiently map a statistical signal processing-based OMP framework for SNS spectrum recovery on Zynq SoC (ZSoC). ZSoC is the heterogeneous SoC comprising of software (SW), that is, dual-core Cortex ARM A9 processor, and hardware (HW), that is, seven-series field-programmable gate array (FPGA). To address the drawbacks of OMP and make the spectrum reconstruction agnostic to sparsity, we replace OMP with a convolutional neural network (CNN)-based DL architecture and efficiently map it on the ZSoC platform as the second contribution. Via hardware–software codesign (HSCD), we explore different versions of the proposed DL architecture obtained by partitioning between SW (ARM processor) and HW (FPGA). Our study offers exciting insights which may not be visible in conventional theoretical- and simulation-based analyses. For these architectures, we present the resource utilization, power consumption, and execution time comparisons for given memory constraints and a wide range of word lengths (WLs) at the parameters and computation levels. We develop an end-to-end application with a live graphical user interface (GUI) to demonstrate the real-time WSS on the ZSoC platform. Refer to [28] for source codes, HW IPs, datasets, and detailed tutorial used for generating the results presented in this article.

The rest of this article is organized as follows. We discuss the OMP architecture and experimental results in Section II. The DLWSS algorithm is presented in Section III and the corresponding architecture in Section IV. Section V presents the performance analysis results and comparison of DLWSS with OMP followed by complexity analysis in Section VI. Section VII concludes this article.

II. SPECTRUM RECOVERY VIA OMP FOR SNS-BASED WSS

This section discusses the design and implementation of the OMP on ZSoC. We demonstrate the various drawbacks of OMP via experimental analysis and results.

A. OMP on ZSoC

OMP [11], [12], [29] is one of the most widely used sparse recovery algorithms for SNS-based WSS. It follows an iterative formulation where an occupied band that is highly correlated with the residual matrix is identified. Then its contribution is removed from the residual matrix to identify the next highly correlated occupied band. This process is repeated for all occupied bands when we have prior knowledge of the spectrum sparsity. Otherwise, we need to have a stopping criterion based on estimated sparsity. The OMP pseudocode is given in Algorithm 1. Here, \( K \) represents the number of ADCs used for SNS, \( Q \) is the number of snapshots produced by the ADC, and \( N \) is the number of frequency bands in the wideband spectrum. Furthermore, \( A \) is the sensing matrix of dimension \( K \times N \), \( Y \) are the received SNS samples of dimension \( N \times Q \), and \( \dagger \) represents a matrix pseudoinverse. It comprises four main steps: 1) matching (lines 6–8); 2) identification (lines 9–13); 3) least squares (line 14); and 4) approximation (line 15).

We have realized the OMP in Algorithm 1 on ZSoC ZC706 comprising of Dual ARM Cortex A9 processor and seven-series FPGA with 1090 units of 18-kB block RAMs (BRAMs), 900 DSP48E units, 218 600 units of six-input look-up tables (LUTs) and 437 200 flip-flops (FFs). Table I shows the execution time, resource utilization, and power consumption comparison for the two best possible realizations of the OMP algorithm: 1) only SW (ARM) and 2) HSCD (ARM + FPGA). Note that we have optimized the code for SW implementation and carefully chosen the WL on HW to minimize the execution time, power consumption, and resource utilization without compromising functional accuracy. The OMP realization on ZSoC outperforms the SW realization with around 3.5 times lower execution time. Although the OMP algorithm is iterative, parallelizing the matrix multiplications and matrix inversions using lower–upper (LU) decomposition results in improved performance. The gain in execution time increases with the increase in matrix size. The use of FPGA leads to high resource utilization and power consumption.

| Algorithm 1 OMP Algorithm for WSS |
|-----------------------------------|
| **Require**: Sensing matrix \( A[K][N] \), aliased sub-Nyquist samples \( Y[K][Q] \), Sparsity \( S \) |
| 1. \( \text{occupied\_bands} \leftarrow [\] |
| 2. \( A_{\text{norm}} \leftarrow \text{Column normalized } A \) \#Normalize A |
| 3. \( \text{Res} \leftarrow Y \) \#Initialize Residual |
| 4. \( \text{iter} \leftarrow 1 \) |
| 5. while \( \text{iter} \leq S \) do \#No. of occupied bands |
| 6. for \( j \) in columns\((A_{\text{norm}})\) do |
| 7. \( Z[j] \leftarrow \text{norm}(A_{\text{norm}}[:,j])^{T} \times \text{Res} \) |
| 8. end for |
| 9. Append \arg\max(Z) to occupied\_bands |
| 10. for \( j \) in occupied\_bands do |
| 11. \( \text{Append column } A_{\text{norm}}[:,j] \) to As |
| 12. end for |
| 13. \( \text{Res} \leftarrow \text{Res} - A_{\text{norm}} \times (A_{\text{norm}}^{T} \times \text{Res}) \) |
| 14. \( \text{iter} \leftarrow \text{iter} + 1 \) |
| 15. end while |
| 16. return occupied\_bands |
B. Functional Performance Analysis of OMP on ZSoC

In this section, we analyze the functional correctness of the OMP algorithm realized on ZSoC. To begin with, we discuss the performance metrics used for such analysis.

1) Performance Metrics: We consider two well-known performance metrics: detection accuracy of all bands in percentage ($P_d^{AB}$) and detection accuracy of occupied bands in percentage ($P_d^{OB}$) [30], [31]. The metric, $P_d^{AB}$, corresponds to the fraction of frequency bands whose status is correctly detected by the algorithm. The second metric, $P_d^{OB}$, corresponds to the fraction of occupied bands whose status is correctly detected by the algorithm. Mathematically, $P_d^{AB}$ and $P_d^{OB}$ are formulated as

\[
P_d^{AB} = 100 \times \frac{\sum_{n=1}^{N} (y_{\text{pred}}^n = y_{\text{true}}^n)}{\sum_{n=1}^{N} (y_{\text{true}}^n)}
\]

\[
P_d^{OB} = 100 \times \frac{\sum_{n=1}^{N} (y_{\text{pred}}^n = 1 \& y_{\text{true}}^n = 1)}{\sum_{n=1}^{N} (y_{\text{true}}^n = 1)}
\]

Here, $N$ is the total number of frequency bands in the digitized spectrum, $y_{\text{pred}}$ and $y_{\text{true}}$ are the predicted and ground-truth band status, respectively (i.e., 0 for vacant and 1 for the occupied band). It is worth noting that $P_d^{OB}$ is the preferred metric for a sparse spectrum consisting of fewer occupied bands since $P_d^{AB}$ may give a high value even when the algorithm erroneously detects all bands as vacant. Similarly, $P_d^{AB}$ is the preferred metric for spectrum with lower sparsity comprising the higher number of occupied bands. For such a spectrum, $P_d^{OB}$ may be high even if the algorithm erroneously detects all bands as occupied. We do not consider the spectrum recovery error as a performance metric since it does not offer additional insights for WSS. There are challenges in capturing a large number of samples from HW for accurate error calculation of spectrum recovery error in real-time.

2) Dataset: We consider the SNS with $K = 8$ ADCs, which allow the recovery of the spectrum when the number of occupied bands is less than or equal to $K$. Since SNS-based WSS is based on the underlining assumption of the sparse spectrum, we have generated various datasets with sparsity ranging from 50% to 100%. Hence, we have selected the total number of bands, $N$, as 14. We have fixed $Q$ as 299.

For DLWSS training, the signal-to-noise ratio (SNR) range is $-20$ to $10$ dB with a step size of 2. We have selected 7000 samples of each SNR with an 80:20 split between training and validation datasets. Thus, the total dataset size is 112000 samples with 89600 and 22400 samples for training and validation, respectively.

The dataset used for testing OMP and DLWSS algorithms consists of 1400 samples of each SNR, that is, 22400 samples. For better analysis, we consider two groups of datasets for testing based on sparsity level. Note that the training is done on a common dataset. These two groups, each of size 11200 samples, are as follows.

1) Extremely sparse spectrum (ESS): This dataset contains the spectrum with the number of occupied bands between 1 and 3.

2) Highly sparse spectrum (HSS): This dataset contains the spectrum with the number of occupied bands between 4 and 7.

As discussed before, we use $P_d^{OB}$ and $P_d^{AB}$ as performance metrics for ESS and HSS datasets, respectively. Interested readers can refer to [28] for datasets and source code used for the dataset generation.

3) OMP Without Sparsity Knowledge: As discussed in Algorithm 1, the number of iterations in the OMP algorithm depends on the sparsity, $S$ (line 5) and this requirement limits its usefulness in realistic applications. We consider the variation of OMP where sparsity knowledge is not known by exploring the stopping criteria for residual, such as $||\text{Res}|| < \epsilon$, where $\epsilon$ is the convergence constant. The convergence constant, $\epsilon$, depends on both the sparsity and SNR of the digitized spectrum. To fix $\epsilon$, we assume prior knowledge of the SNR, which is common in wireless systems since wireless receivers can easily measure the SNR of the digitized spectrum. Next, via empirical analysis, we study the correlation between the SNR and residual, $||\text{Res}||$, to get the desired performance. We use this value of $||\text{Res}||$ as an appropriate estimate for $\epsilon$ for a given SNR. In Fig. 1, we compare the desired values of $\epsilon$ for a wide range of spectrum sparsity and SNRs. Note that we cannot have fixed $\epsilon$ for all SNRs. However, variation in $\epsilon$ for different spectrum sparsity is insignificant, and we can use the average value of $\epsilon$ for all spectrums with varying sparsity. We refer to this algorithm as OMP-$\epsilon$.

In a wireless environment, the wireless channel may vary depending on the deployment scenario. In Fig. 2, we study the effect of wireless channel on the average value of $\epsilon$ obtained over a wide range of SNRs, $\epsilon_{\text{SNR}}$. As expected, the effect of the channel on the WSS is limited since we do not need to recover the spectrum. Thus, we can fix a single value of $\epsilon_{\text{SNR}}$ for all types of wireless channels.
4) Performance Analysis and Drawbacks: In Fig. 3, we compare the performance of the OMP and OMP-ε for ESS and HSS datasets for a wide range of SNRs, respectively. As expected, the performance improves with the increase in the SNR. In the case of ESS with $P^\text{OB}_d$ as a performance metric, both OMPs achieve 100% accuracy at a high SNR due to accurate estimation of $\epsilon$. However, the difference between OMP and OMP-ε is around 10%–40% at low SNR. In the case of HSS with $P^\text{AB}_d$ as a performance metric, overall accuracy is lower, and OMP-ε cannot meet the accuracy of OMP even at a high SNR. This is because the accurate selection of $\epsilon_{\text{SNR}}$ guarantees the correct detection of occupied bands but does not guarantee the correct detection of the status of remaining bands. These results indicate that the state-of-the-art OMP algorithm does not offer reliable performance for HSS, even at a high SNR.

OMP suffers from multiple drawbacks. First, prior knowledge of sparsity is needed. If such knowledge is unavailable, it suffers from significant performance degradation even with prior knowledge of the SNR. Second, poor performance at a low SNR for both types of spectrum. Third, poor performance even at a high SNR for HSS. These shortcomings motivate the search for potential alternatives for SNS-based WSS.

III. DLWSS: Spectrum Recovery via DL for SNS-Based WSS

Previous efforts that aim to tackle SNS-based WSS using deep networks can be broadly classified into two categories:

1) iterative approaches [17], [18], [19] that follow the OMP formulation and augment the frequency band status detection with a deep network and 2) noniterative approaches that utilize a deep network for end-to-end spectrum recovery and frequency band status detection [32]. Since the iterative approaches follow the algorithmic formulation of OMP, they suffer from the same limitations as that of OMP. In this article, we focus on a noniterative approach that can handle the signals of varying sparsity and does not require sparsity information or any manually adjusted convergence constant [32]. The proposed DLWSS algorithm aims to learn an end-to-end model for WSS. DLWSS pipeline comprises two stages: 1) preprocessing and 2) deep network prediction.

A. Preprocessing

Algorithm 2 shows the steps involved in the preprocessing stage of the DLWSS. The preprocessing step receives the signal captured by the antenna and digitized it using the SNS-based analog front-end. It processes and normalizes the digitized signal, so the DL architecture can handle it. Specifically, this involves the computation of the pseudorecovered spectrum $\hat{X}$ using the sensing matrix and sub-Nyquist samples. Since the complex input signal $\hat{X}$ cannot be fed directly to the DL architecture, it is converted to a high-dimensional real-valued signal as shown in Algorithm 2. In the end, the signal is normalized for faster convergence of the training process.

Algorithm 2: Preprocessing

Require: Sensing matrix $A[K][N]$, aliased sub-Nyquist samples $Y[K][Q]

1: $A_y = A^\ast \times A$ #Square Matrix $A_y$
2: $P \times A_y = L \times D \times U$ #LU Factorization
3: $A_y^{-1} = U^{-1} \times D^{-1} \times L^{-1} \times P^{-1}$
4: $A^\top = A_y^{-1} \times A^\top$ #Pseudo-inverse of $A$
5: $X \leftarrow A^\top \times Y$
6: $X_d \leftarrow \text{concatenate}(X_{\text{real}}, X_{\text{imag}})$
7: $X_n \leftarrow \text{Normalize}(X_d)$
8: return $X_n$

B. Deep Network Prediction

The deep network block of the DLWSS receives the processed input samples from preprocessing block and predicts the status (vacant or occupied) of each frequency band of the digitized spectrum. The DLWSS architecture is based on a CNN due to its ability to capture spatial correlation in input signals which is integral for the spectrum-sensing task. Furthermore, parameter sharing allows them to operate with fewer parameters, making the network memory efficient and a good candidate for HW realization. Table II shows the architecture of the DL model, which comprises three convolutional (CV) layers and a single fully connected (FC) layer. All intermediate activations are Rectified Linear Units (ReLUs), and the activation at the output layer is Sigmoid.

Similar to the deployment of any DL algorithm, DLWSS design has two phases: training and inference. The training
TABLE II
CNN-BASED DEEP NETWORK PREDICTION ARCHITECTURE

| Layers | Filters | Kernel size | Input Shape | Output Shape |
|--------|---------|-------------|-------------|--------------|
| CV 256 | 1x150  | 14x299x2    | 14x150x256  |
| CV 128 | 1x100  | 14x150x256  | 14x51x128   |
| CV 64  | 1x31   | 14x51x128   | 14x1x64     |
| Flatten| -      | 14x1x64     | 896x1       |
| FC     | -      | -           | 896x1       |

Fig. 4. Ablation study of the DLWSS on (a) ESS and (b) HSS spectrum for different numbers of CV levels.

phase minimizes a loss function, which measures the difference between the predicted and actual labels. Since more than one frequency band can be occupied in a wideband spectrum, we formulate the problem as a multilabel binary classification with binary cross-entropy as the training loss function on final sigmoid outputs. After the training mode, the model weights are frozen, and the CNN model is used in inference mode to find the occupancy status of an unknown signal in real-time. We have trained the architecture for a dataset using a machine-learning framework (PyTorch in our case) and GPUs. After the training and ablation study, model architecture, weights, and parameters are extracted for realization on the ZSoC, followed by inference on the new dataset.

The number of layers, types, and filter sizes in the DLWSS model are selected after an in-depth ablation study. As shown in Fig. 4, we compare the performance of the DLWSS for the different numbers of CV layers. It can be observed that the three-layer model offers superior performance than the two- and four-layer models, especially at low SNRs and ESS datasets. Next, we study the effect of filter sizes on the performance of the three-layer DLWSS model as it significantly impacts the HW resource utilization. As shown in Fig. 5, we consider four different models with filter sizes denoted as $X$, $Y$, $Z$. This corresponds to the filters of size $1 \times X$, $1 \times Y$, and $1 \times Z$ on layers 1, 2, and 3, respectively. It can be observed that the filter size of 150, 100, 51 offers superior performance.

IV. REALIZATION OF DLWSS ON ZSoC

This section presents the complete architecture and mapping of DLWSS on ZSoC and real-time demonstration via a Linux-based GUI deployed on ZSoC. In Fig. 6, various building blocks of the DLWSS, such as Preprocessing, CNN and FC layers, scheduler, direct-memory access (DMA), interrupt, and GUI controller, are shown. For illustration, we have shown the Preprocessing, CNN, and FC layer processing on FPGA. To enable this, we have developed AXI-stream-compatible HW IPs for Preprocessing, CNN, and FC blocks and interconnected them with SW via DMA in the scatter-gather mode for efficient data transfers. Later in Section VI, we considered various architectures via HSCD by moving the blocks between the ARM Processor and FPGA. We have deployed a Linux-based operating system on SW, which takes care of various scheduling and controlling operations. It also enables GUI development for real-time demonstration.

As discussed in Algorithm 2, Preprocessing stage involves large-size matrix multiplication and inversion operations. We have modified Xilinx’s matrix multiplication and matrix inversion reference examples to support the complex number arithmetic since the baseband wireless spectrum is represented using complex samples. The well-known LU decomposition method is selected for matrix inversion. We have included data buffers using block memory to minimize the repeated data communication between SW and HW and enable matrix operations of different sizes. While the overall algorithm is sequential, we parallelize individual operations like Matrix Conjugate Transpose and Matrix Multiplication on FPGA. Every element in the matrix is parallely processed to compute transpose, and every row column dot product in matrix multiplication is performed in parallel to speed up the computation. In addition, multiple instances of these IPs are integrated to get the desired preprocessing functionality, as shown in Fig. 6. In the end, a normalization block is included to meet the input requirements of a deep neural network.

Next, we focus on mapping each CNN layer on the FPGA. As shown in Fig. 6, the CNN involves many multiply and accumulation operations on noncontiguous data, that is, frequent reading and writing from memory is needed. Depending on the CNN layer dimensions, it may not be possible to store all weights, and input data in on-chip memory such as BRAM on FPGA due to limited size and fewer read/write ports. For instance, if we store all the inputs and weights of the CNN model considered in Table II in the BRAM with single-precision floating (SPFL) number representation, we need a total of 116.4 megabyte (MB) of BRAM, assuming...
the CNN output is written directly in the external memory. Such a large amount of on-chip memory is expensive and may not be available in most SoC. Using external memory leads to frequent data communication overhead resulting in high latency. Thus, mapping the CNN layer on FPGA requires careful data sharing between external and on-chip memory to get the desired latency. Underlining architecture can be layer-specific depending on the various parameters of the CNN layer, resource, and latency constraints.

For the CNN model in Table II, we have explored the memory tiling approach in which small tiles or blocks of weights and inputs are loaded into the on-chip memory and care has been taken to maximize the utilization of data currently present in on-chip memory. We define four parameters, \(T_o, T_i, T_r, T_c\), that is, the tiling factors of output channels, input channels, output rows, and output columns, respectively. For easier understanding, we present an illustrative example with tiling parameters \(1, 2, 2, 2\) in Fig. 7. The input is of size \(4 \times 4 \times 2\), that is, two channels and it is convoluted with two filters of dimension \(2 \times 2 \times 2\). Initially, we initialize an output tile and load the input tile and weight tile required to compute the output tile into the on-chip BRAM. For instance, to compute a \(2 \times 2 \times 1\) output tile, we need a \(3 \times 3 \times 2\) input tile \((I_{\text{tile}})\) and one filter of dimension \(2 \times 2 \times 2\) \((W_{\text{tile}})\). After loading, the tiled convolution is performed between \(I_{\text{tile}}\) and \(W_{\text{tile}}\) using a set of parallel multipliers and adders, referred to as CNN computations in Fig. 7. Once all elements of the output tile are computed, the tile is sent back to the external memory, and a similar process is repeated for computing the next output tile. Here, we have shown the computation of one element of the output tile. Depending on resource availability, latency constraints, and memory ports, we can have CNN computations of all elements of a tile or even multiple tiles in a parallel or serial–parallel fashion.

The tiling approach significantly reduces the on-chip memory requirement for the CNN model in Table II. For instance, tiling with parameters \(20, 16, 20, 20\) needs only 3.35 MB of BRAM (0.24 MB for the output tile, 1.464 MB for the weight tile, and 1.650 MB for the input tile) compared to 116.4 MB in nontiling-based architecture. This in turn allows efficient optimization of HW IP cores via pipelining and unrolling, resulting in significant improvement in performance. We have...
explored a wide range of tiling parameters and implemented these architectures on the ZSoC. Refer to Section VI-C for more details. We have explored a similar tiling approach for FC layers as well. However, experimental analysis shows that FC layers do not need tiling due to the smaller dimensions of inputs and weights.

The DLWSS model in Table II contains two types of non-linear activations: ReLU and Sigmoid. The realization of the ReLU on the FPGA is simple as it needs only one comparator and multiplexer. The realization of the Sigmoid on the FPGA can be done either using the LUT-based approach or by polynomial approximation. The LUT-based approach is memory-intensive, while the polynomial-based approach involves many arithmetic and logical operations. Based on the experimental analysis, we have realized ReLU on FPGA and Sigmoid on the ARM processor.

Using the proposed architectures, an end-to-end application with a live GUI is developed to demonstrate the real-time WSS on the ZSoC platform. The application running on a Petalinux-based operating system deployed on the ARM processor accelerates the computation of preprocessing algorithm and the DL model on the FPGA. The application obtains and stores real-time predictions on the SD card. The GUI application, shown in Fig. 8, is deployed on the remote server and reads the contents of the SD card at regular intervals. The GUI is developed using the Tkinter framework and provides visualization in real-time as and when the architecture predicts the status of frequency bands in the received digitized spectrum.

V. PERFORMANCE ANALYSIS AND COMPARISON WITH OMP

This section compares the functionality of the OMP and DLWSS architectures for different wireless channels, a wide range of SNRs, and sparsity levels. We consider the effect of prior knowledge of sparsity on performance. As discussed in Section II-B, we use $P_d^{\text{ABS}}$ and $P_d^{\text{AB}}$ as the performance metrics for ESS and HSS spectrums, respectively. We consider the floating-point arithmetic-based architecture realized on the ZSoC platform for all the experimental results presented in this section.

A. Accuracy Comparison

In Fig. 9, we compare the performance of OMP and DLWSS for three different channels (AWGN, Rayleigh, and Rician). We assume the prior knowledge of the spectrum sparsity in the case of OMP. For the ESS spectrum [see Fig. 9(a)], the DLWSS performs better than OMP with an average performance gain of around 12%. We also observed that DLWSS and OMP are robust to changes in channel conditions, which is expected since channel conditions’ impact on spectrum sensing is fairly limited. For the HSS spectrum [see Fig. 9(b)], the DLWSS significantly outperforms the OMP algorithm with an average performance improvement of around 24%.

In practical deployments, spectrum occupancy changes dynamically with time; hence, knowledge of spectrum sparsity is unavailable. In Fig. 10, we analyze the DLWSS and OMP algorithms’ performance when spectrum sparsity is unknown. It is observed that the proposed DLWSS significantly outperforms the OMP algorithm at all ranges of SNRs, with an average improvement of around 36.4% and 31.8% for ESS and HSS datasets, respectively.

In Fig. 11, we compare the impact of sparsity on the performance of DLWSS and OMP algorithms when spectrum sparsity is unknown. We consider two SNRs: 0 and 10 dB. In Fig. 11, $\text{OMP}_{\text{XdB}}$ and $\text{DLWSS}_{\text{XdB}}$ correspond to SNR of X dB. We trained the DLWSS model using the dataset comprising an equal number of samples from each sparsity level. It can be observed that the DLWSS model using the dataset comprising an equal number of samples from each sparsity level. It can be observed that the DLWSS model using the dataset comprising an equal number of samples from each sparsity level.
TABLE III
HSCD Study of DLWSS Architecture on ZSoC

| No. | Blocks on HW | Blocks on SW | Execution Time (s) | (BRAM, DSP, FF, LUT) | (Total Power, Dynamic Power)(Watts) |
|-----|--------------|--------------|-------------------|----------------------|-------------------------------------|
| 1   | P+CV+FC+A    |              | 30.8              | (0, 0, 0, 0)         | (1.6, 1.2)                          |
| 2   | CV           | P+FC+A       | 2.87              | (220, 719, 126741, 100250) | (3.205, 2.205)                       |
| 3   | CV+FC        | P+A          | 2.869             | (249, 724, 167676, 117382) | (3.373, 2.369)                       |
| 4   | P+CV         | FC+A         | 2.863             | (296, 818, 139857, 117738) | (3.391, 2.387)                       |
| 5   | P+CV+FC      | A            | 2.863             | (325, 823, 180792, 134870) | (3.540, 2.533)                       |
| 6   | P+CV+FC+A    |              | 2.864             | (325, 853, 183173, 138230) | (3.567, 2.559)                       |

Table III shows the results of our HSCD study for the DLWSS architecture comprising preprocessing (P), CV, FC, and activations (A) blocks. As shown in Row 1, mapping the entire architecture on SW results in a high execution time of 30.8 s. As we shift more blocks to HW, the execution time is reduced while resource and power consumption increase (Rows 2–6). The CV layer is the most computationally complex unit in the DLWSS, and realizing it on the HW results in a significant reduction in the execution time from 30.8 to 2.87 s.

Since the DLWSS model has fewer FC layers than the CV layers, shifting the FC layer to HW does not offer substantial improvement in execution time compared to 2.7%, 0.6%, 9.4%, and 7.8% increase in BRAM, DSP, FF, and LUT utilization, respectively. Similarly, shifting the preprocessing algorithm (P) on HW leads to an improvement in execution time of around 7 ms compared to an increase in the BRAM, DSP, and LUT usage by 7%, 11%, and 8%, respectively, with respect to previous architecture in Row 3. As we move the FC block to HW (Row 5), we notice that the impact of the FC block is the same as seen in Row 2, which is expected as adding the FC block adds constant time and resource utilization for all cases. In Row 6, we move the Sigmoid-based activation block to HW, which increases the DSP and LUT utilization by 3% and 1.5%, respectively, with around 1-ms improvement in execution time. As we gradually shift more blocks from SW to HW, the overall power consumption increases from 3.205 (Row 2) to 3.567 W (Row 6). Thus, from the HSCD perspective, it is better to keep all blocks except CV layers on the SW, given the high resource penalty in HW for a small gain in execution time.

Thus, results in Figs. 9–11 confirm the superiority of the DLWSS over the OMP-based approach.

VI. COMPLEXITY ANALYSIS
In this section, we analyze the complexity of the DLWSS algorithms for different architectures realized via HSCD, various WLs, and memory tiling approaches.

A. Hardware Software Codesign
A heterogeneous SoC such as ZSoC from Xilinx contains ARM Cores as the processing system (SW) and FPGA as the programmable logic (HW). An integral aspect of developing an efficient mapping of the DLWSS on the ZSoC is to design an optimal HSCD strategy to facilitate functionally accurate architecture for the desired latency and resource constraints. Specifically, we need to decide how to partition the algorithm between HW and SW and minimize the data communication overhead between them. The HSCD is important since sequential operations, scheduling tasks, and GUI are preferred on SW. At the same time, FPGA can efficiently handle the task, which can be accelerated via parallel processing. However, in certain situations, serial tasks are preferred on HW, while parallel tasks are preferred on SW to avoid data communication overhead between SW and HW. Furthermore, some operations may offer speed up on HW. Still, such speed-up may not be significant compared to the algorithm’s overall execution time; hence, realizing such tasks on SW can reduce FPGA size, cost, and power consumption. Such tradeoffs demand a detailed study of various HSCD architectures to design an architecture that meets the cost, latency, and power constraints.

B. WL Optimization
Conventionally, HW realization of the algorithm in floating-point arithmetic offers good functional accuracy but incurs high resource utilization, power consumption, and execution time. Since the extremely large dynamic range offered by floating-point arithmetic may not be needed for all the subblocks of the algorithm, fixed-point arithmetic can potentially offer a significant reduction in resource, power, and execution time without compromising functional accuracy. In wireless applications such as DLWSS, the dynamic range of inputs, weights, and activation is limited due to ADCs; hence, fixed-point architectures are preferred. In this section, we discuss the selection of appropriate WL for the part of the algorithm realized on the HW, that is, FPGA, and its impact on functional accuracy, resource utilization, execution time, and power consumption.
The WL optimization can be done during and after training. During training, recent frameworks such as HLS4ML [34], QKeras [35], and Xilinx FINN [36] can be explored. After training, the WL optimization is done manually based on the range of inputs, outputs, and intermediate signals. In each case, the aim is to lower WL without compromising functional accuracy. We have explored these approaches and presented the results corresponding to the second approach, where WL optimization is done on the trained model with floating-point WL. This is because it offered superior performance than the HLS4ML- and QKeras-based quantized models. Nevertheless, the proposed architecture can be tuned to any desired WL, irrespective of whether it is chosen during or after training.

We use $W$ bits to represent each number in fixed-point quantization. Out of $W$ bits, we use $I$ bits to represent the integer part and $(W-I)$ bits to represent the fractional part. For example, the fixed-point representation of the number 3.25 needs only six bits with $I = 3$ compared to 64 bits in double-precision floating point (DPFP), 32 bits in single-precision floating point (SPFP), and 16 bits in half-precision floating point (HPFP). Thus, depending on the dynamic range of the given variable, the appropriate selection of $W$ and $I$ can avoid a loss in functional accuracy. To identify appropriate values of $W$ and $I$ for the DLWSS architecture, we analyzed various subblocks’ dynamic range of inputs, outputs, and intermediate outputs. For instance, Table IV shows the analysis to determine the optimal integer width for HW realization of CV and FC layers of the DLWSS, where we infer the model on samples of the dataset to estimate the ranges of intermediate activations and the model weights.

It can be observed that the minimum value of $I$ is 9 and 2 bits for activation and weights, respectively. The value of $W$ depends on the number of bits for accurate fractional number representation to get the desired functional performance, and we select them via heuristic experiments.

We have designed and implemented DLWSS architectures of various WLs on ZSoC and analyzed their performance to identify appropriate WL for fractional number representation. In Table V, we compare ten different DLWSS architectures with fixed integer WL of 9 for activation and 2 for weights. Here, we have fixed the WL of weights to (16, 2) to identify the WL for activation. A similar process is done to identify the WL of weights by fixing the WL of activation. Corresponding details are omitted to avoid repetition of discussion. All these details are given in Rows 3–10 of Table V. As expected, there is a slight degradation in functional accuracy as WL decreases. The DLWSS architecture with the WL of (23, 9) or below in Rows 9 and 10 suffers from significant degradation in performance and should be avoided. The DLWSS architecture in Row 7 with a WL of 25 offers functional performance the same as that of floating-point architectures with more than 50% savings in DSP, FFs, and LUTs over HPFP architecture in Row 2 and over 60% savings in DSP, FFs, and LUTs over SPFP architecture in Row 1. These savings can be further improved using the DLWSS architecture with WL of (24, 9) in Row 8 with minor degradation in performance. Note that we have assumed that the inputs and outputs are in SPFP format; hence, additional WL conversion inside the architecture is needed. We can reduce the execution time further if the input and output WLs are the same as the rest of the architectures.

In Table V, we have fixed the SNR to 10 dB. To analyze the architecture’s performance at different SNRs, we have compared the $P_{dB}$ for ESS and $P_{dB}$ for HSS for SNRs ranging from −20 to 10 dB in Fig. 12. It can be observed that the performance degrades at higher SNR due to insufficient WL.

| Type | Layer | Minimum Value | Maximum Value | $I_{min}$ |
|------|-------|---------------|---------------|-----------|
| Activation | CV | -77,061 | 199,309 | 9 |
| Activation | FC | -86,594 | 158,975 | 9 |
| Weight | CV | -0.4812 | 0.9561 | 2 |
| Weight | FC | -0.0661 | 0.0219 | 2 |

**Table IV**

**Integer WL Selection for DLWSS**

![Fig. 12. Impact of the WL on the functional accuracy of the DLWSS architecture.](image)

Fig. 12. Impact of the WL on the functional accuracy of the DLWSS architecture.

**TABLE V**

| SNR (dB) | $P_{dB}$ for ESS | $P_{dB}$ for HSS |
|---------|------------------|------------------|
| -20     | 50               | 50               |
| -10     | 60               | 60               |
| 0       | 70               | 70               |
| 10      | 80               | 80               |

**C. Impact of Memory Tiling Approach**

As discussed in Section IV, memory tiling is essential to enable an efficient implementation of the memory-intensive CV layer operations on HW. In this section, we discuss the impact of tiling size on the performance of the DLWSS architecture realized on the SW and HW. In Fig. 13, we compare the acceleration factor, that is, the ratio of execution time on SW and HW, for different tiling sizes on SW and HW realizations. It can be observed that the acceleration factor increases with the increase in the tiling size of the HW architecture for a given tiling size of SW realization. Such behavior can be attributed to the following.
TABLE V
FUNCTIONALITY AND COMPLEXITY ANALYSIS OF DLWSS ARCHITECTURES FOR DIFFERENT WLs AT 10-dB SNR

| No. | Activation WL: <W_a, L_a> | Weights WL: <W_l, L_u> | Execution Time (s) | ESS: P^{1b}_d | HSS: P^{AB}_d | (BRAM, DSP, FF, LUT) | (Tot. Power, Dyn. Power) |
|-----|--------------------------|-----------------------|-------------------|--------------|-------------|---------------------|--------------------------|
| 1   | SPFP                     | SPFP                  | 2.28              | 100          | 100         | [256,879,150107,117023] | (4,218,3,944)           |
| 2   | HPFP                     | HPFP                  | 2.11              | 100          | 99.97      | (213,719,98614,74952) | (2,688,2,448)           |
| 3   | <29.9,>                  | <16,2>               | 2.28              | 100          | 100        | [248,399,75935,81119] | (2,523,2,285)           |
| 4   | <28.9,>                  | <16,2>               | 2.28              | 100          | 99.98      | (248,399,71907,74381) | (2,484,2,247)           |
| 5   | <27.9,>                  | <16,2>               | 2.30              | 100          | 99.94      | (243,399,59641,50259) | (2,353,2,119)           |
| 6   | <26.9,>                  | <16,2>               | 2.30              | 100          | 99.94      | (240,399,59641,50259) | (2,351,2,114)           |
| 7   | <25.9,>                  | <16,2>               | 2.30              | 100          | 99.94      | (240,399,53926,14443) | (2,294,2,065)           |
| 8   | <24.9,>                  | <16,2>               | 2.30              | 100          | 98.78      | (232,399,53926,14443) | (2,292,2,061)           |
| 9   | <23.9,>                  | <16,2>               | 2.28              | 87.38        | 79.45      | (232,399,53926,14443) | (2,292,2,061)           |
| 10  | <22.9,>                  | <16,2>               | 2.28              | 70.96        | 57.46      | (232,399,53926,14443) | (2,292,2,061)           |

1) Higher tile size on HW means fewer accesses to external memory since more data is buffered on-chip BRAM.
2) Higher tile size allows more opportunities for intratile parallelization, that is, dot products within a tile can be computed in parallel on HW.

Interestingly, an increase in the tiling size of the SW realization results in a degradation in the execution time of the DLWSS on SW. For instance, the acceleration factor is higher for SW tiling size of (20, 16, 20, 20) compared to SW tiling size of (2, 2, 2, 2). This behavior is reversed compared to HW, and one possible reason is data caching in SW realization. Smaller tile size allows tiles to be cached in the local data cache memory of ARM cores, thereby limiting the number of accesses to external memory. Since cache size is significantly smaller than on-chip BRAM on HW, larger tile size results in frequent cache flush requirements, resulting in execution time degradation.

Though the tiling size does not impact functional accuracy, a larger tiling size leads to higher resource utilization, as shown in Table VI. This is because a larger tile size needs a higher amount of on-chip memory and enables parallel computations due to the availability of more data on the chip. The use of fixed-point arithmetic can help to reduce resource utilization significantly. Thus, combining HSCD, WL, and tiling parameters is vital to meet the given resource and execution time constraints.

D. Complexity Comparison Between OMP and DLWSS

As discussed in Section V, the DLWSS offers 30%–36% improvement in functional accuracy over OMP. As expected, the execution time, resource utilization, and power consumption of the DLWSS are higher than that of the OMP due to the three-CV layer. The corresponding results are shown in Table VII. This is the penalty paid to overcome all three drawbacks of the OMP algorithms discussed in Section II-B4, along with higher accuracy. Such higher execution time is acceptable in wireless networks where base stations do wideband sensing at regular intervals to gather the spectrum’s occupancy, and spectrum allocation is done based on historical data. Conventional narrowband sensing can be used for applications where spectrum occupancy estimation is needed for immediate spectrum allocation since OMP does not offer sufficient accuracy to minimize interference to legacy users.

The execution time of the DLWSS can be reduced by using the SoC with large on-chip memory and exploring PCI interface-based high-speed communication between the processor and FPGA. In addition, recent advances in neural architecture search (NAS) and quantized model training can potentially reduce the complexity of DLWSS architecture.
VII. CONCLUSION AND FUTURE WORKS

In this article, we designed and implemented statistical OMP- and DL-based algorithms on ZSoC for wideband sensing applications. We have provided in-depth experimental results and complexity comparisons among various architectures obtained via HSCD, WL optimization, and memory tiling. Specifically, we demonstrated the drawbacks of conventional OMP algorithms, such as poor performance at a low SNR and the need for prior knowledge of sparsity. These drawbacks are addressed via a novel DL-based approach. However, the DL architecture’s high resource utilization and execution time is a concern. In the future, we also plan to explore NAS along with quantized model training to reduce the execution time of DL architecture. We also plan to integrate the proposed architecture with analog-front-end for experiments with real-radio signals.

REFERENCES

[1] A. Ali and W. Hamouda, “Advances on spectrum sensing for cognitive radio networks: Theory and applications,” IEEE Commun. Surveys Tuts., vol. 19, no. 2, pp. 1277–1304, 2nd Quart., 2017.
[2] C.-P. Yen, Y. Tsai, and X. D. Wang, “Wideband spectrum sensing based on sub-Nyquist sampling,” IEEE Trans. Signal Process., vol. 61, no. 12, pp. 3028–3040, Jun. 2013.
[3] M. Labib, V. Marojevic, J. H. Reed, and A. I. Zaghoul, “Extending LTE into the unlicensed spectrum: Technical analysis of the proposed variants,” IEEE Commun. Standards Mag., vol. 1, no. 4, pp. 31–39, Dec. 2017.
[4] W. S. H. M. W. Ahmad et al., “5G technology: Towards dynamic spectrum sharing using cognitive radio networks,” IEEE Access, vol. 8, pp. 14460–14488, 2020.
[5] L. Zhang, M. Xiao, G. Wu, M. Alam, Y.-C. Liang, and S. Li, “A survey of advanced techniques for spectrum sharing in 5G networks,” IEEE Wireless Commun., vol. 24, no. 5, pp. 44–51, Oct. 2017.
[6] M. Matinmikko-Blue, S. Yrjölä, V. Seppänen, P. Ahokangas, H. Hammäinen, and M. Latva-Aho, “Analysis of spectrum valuation elements for local 5G networks: Case study of 3.5-GHz band,” IEEE Trans. Cogn. Commun. Netw., vol. 5, no. 3, pp. 741–753, Sep. 2019.
[7] V. Valenta, R. Marsálek, G. Baudoín, M. Villegas, M. Suárez, and F. Robert, “Survey on spectrum utilization in Europe: Measurements, analyses and observations,” in Proc. 5th Int. ICST Conf. Cogn. Radio Networks: A comprehensive survey,” IEEE Trans. Commun., vol. 19, no. 4, pp. 31–39, Apr. 2016.
[8] Y. Chen and H. S. Oh, “A survey of measurement-based spectrum occupancy modeling for cognitive radios,” IEEE Commun. Surveys Tuts., vol. 18, no. 1, pp. 848–859, 1st Quart., 2016.
[9] M. López-Benitez and F. Casadevall, “Space–dimension models of spectrum usage for cognitive radio networks,” IEEE Trans. Veh. Technol., vol. 66, no. 1, pp. 306–320, Jan. 2017.
[10] S. S. Ioushua, O. Yair, D. Cohen, and Y. C. Eldar, “CaSCADE: Compressed carrier and DOA estimation,” IEEE Trans. Signal Process., vol. 65, no. 10, pp. 2645–2658, May 2017.
[11] E. C. Marques, N. Maciel, L. Naviner, H. Cai, and J. Yang, “A review of sparse recovery algorithms,” IEEE Access, vol. 7, pp. 1300–1322, 2019.
[12] T. T. Cai and L. Wang, “Orthogonal matching pursuit for sparse signal recovery with noise,” IEEE Trans. Inf. Theory, vol. 57, no. 7, pp. 4680–4688, Jul. 2011.
[13] M. Yang and F. de Hoog, “Orthogonal matching pursuit with thresholding and its application in compressive sensing,” IEEE Trans. Signal Process., vol. 63, no. 20, pp. 5479–5486, Oct. 2015.
[14] S. Zhang, J. Wu, D. Chen, S. Li, B. Yu, and J. Qu, “Fast frequency-domain compressed sensing analysis for high-density super-resolution imaging using orthogonal matching pursuit,” IEEE Photon. J., vol. 11, no. 1, pp. 1–8, Feb. 2019.