SCCTS Transactions on Reconfigurable Computing, ISSN: 3049-1533 Vol. 3, No. 2, 2026 (pp. 49-55)
RESEARCH ARTICLE ECEJOURNALS.IN

High-Level Synthesis-Driven Hardware/Software Co-Design for Reconfigurable Embedded AI Accelerators

Shaik Sadulla[1]*, K P Uvarajan[2]

[1]Department of Electronics and Communication Engineering, KKR & KSR Institute of Technology and Sciences, Vinjanampadu, Guntur, Andhra Pradesh, India.

[2]Department of Electronics and Communication Engineering, KSR College of Engineering, Tiruchengode.

Keywords:
High-Level Synthesis (HLS),
Hardware/Software Co-Design,
Reconfigurable Computing,
FPGA,
Embedded AI,
Edge Intelligence,
AI Accelerators,
Energy Efficiency

Author's Email:
sadulla09@gmail.com
Uvarajan@ksrce.ac.in

DOI: 10.31838/RCC/03.02.06

Received : 07.01.2026

Revised : 12.03.2026

Accepted : 08.05.2026

Abstract

High performance and low energy consumption computing devices are necessitated by the increasing global demand of embedded incorporated artificial intelligence (AI) for edge and low power devices. A reconfigurable computing architecture has desirable properties with much interest in using field-programmable gate arrays (FPGA), which support flexible architecture to allow faster processing of AI workloads. Nevertheless, the complexity and time limits are the problems of a traditional RTL-based development that stands in the way of rapid deployment. The paper suggests a streamlined high-level synthesis (HLS) driven hardware / software co-design approach to the design of reconfigurable AI accelerators in the context of embedded platforms. We use HLS tools to synthesize hardware automatically using high-level descriptions and also using performance-guided partitioning functionality to distribute the computations to use both hardware and software. In this paper, we realize and assess AI models, e.g. convolutional neural networks (CNNs) and recurrent neural networks (RNNs) on FPGA-based system-on-chip platforms. It is experimentally shown that the proposed architecture can provide a 5.2-fold speedup with 3.8-fold energy savings on a par with typical CPU-only systems with very little sacrifice in the model accuracy. The suggested framework delivers the potential to offer scalable, effective, and fast producing of AI applications in resource-scarce settings, helping to both prototype and roll out real-time edge cognizance. The work can express the future of HLS-guided co-design as a way to reduce the complexity of development and yet provide substantial performance and energy gains within embedded AI systems.

How to cite this article: Sadulla S, Uvarajan K P (2026). High-Level Synthesis-Driven Hardware/Software Co-Design for Reconfigurable Embedded AI Accelerators. SCCTS Transactions on Reconfigurable Computing, Vol. 3, No. 2, 2026, 49-55

Introduction

With the growing combination of artificial intelligence (AI) with edge computing and embedded platforms, pressure has mounted to develop hardware developed to offer both high computational throughput and low energy consumption. Examples of applications include real-time object detectors, speech recognition, and predictive maintenance, which utilize AI models: specifically convolutional neural networks (CNNs) and recurrent neural networks (RNNs) models, which are computationally and memory demanding.

The performance and energy efficiency demands of these resource-constrained environments is usually not met by the traditional general-purpose processors. The reconfigurable computing platform in the form of field-programmable gate arrays (FPGAs) has proven to offer a viable solution to the above-mentioned issues, since it has the benefit of being architecturally flexible, paralleled and customizable in nature. They permit the insertion of domain-specific accelerators, specific to individual AI loads and deliver significant performance benefits in terms of throughput and power efficiency. Nevertheless, the architecture and compilation of such accelerators in register-transfer level (RTL) form describe complex time-consuming development procedures that slow down fast prototyping and execution.

High-Level Synthesis (HLS) provides an even higher level of abstraction in that the hardware designers are free to specify functionality with high-level programming languages (C/C++ or SystemC). The design cycle is then speeded up and productivity increased as HLS tools automatically generate synthesizable RTL. Combined with a hardware/software (HW/SW) co-design approach HLS enables an efficient partitioning of AI workloads where compute-intensive functions are ported to reconfigurable logic and tasks that require software control are left as software. Although the state of the art has improved over the past, current literature tends to miss a generic framework on handling HLS-driven co-design that tackles the issues of performance optimization, hardware/software division, and hardware utilization in embedded AI applications in a systematic way. In addition, there is little research on running full-fledged AI flow, including inference of CNNs and RNN, on embedded FPGAs with HLS tools and automated co-design flows.

In the present paper, we have tried to bridge this gap, as described in the co-design approach explored in this paper. These are our contributions:

- A custom accelerator generation design flow with an HLS kernels representation of AI;
- The HW/SW partitioning plan is performance driven;
- Benchmark AI model experimental assessment on FPGA system-on-chip (SoC) running platforms.

We show that the development time of our methodology is significantly less than the conventional one but attains the performance up to 5.2x speedup and energy savings of 3.8x compared to the conventional CPU based execution. This paper gives us a feasible roadmap to having efficient and flexible AI that fits in the edge.

Recent appropriate effort has started on HLS to allow AI acceleration on reconfigurable platforms, usually in isolation, lacking a full co-design environment .[1] Our work superiorizes and advances the above efforts to cover end to end HW/SW integration of embedded applications.

Background and Related Work

Embedded AI Workloads

Popular edge devices employing embedded AI workloads are convolutional neural networks (CNNs), recurrent neural networks (RNNs), and transformers to support real-time image classification, natural language processing and sensor data aggregation. Such models are based on a lot of matrix multiplications and convolution, and sequential processing of data, which requires high throughput and memory bandwidth. But the limitations on the energy budget, the computer cache size as well as the thermal constraints of the embedded systems present serious challenges in efficiently executing programs on these systems.[1, 2]

Reconfigurable Computing Platforms

Reconfigurable computing, especially through field-programmable gate arrays (FPGAs) can be viewed as an efficient method toward addressing the computational needs of AI in an embedded context (in terms of performance and energy). In modern FPGAs, there are heterogeneous components like DSP slices, block RAMs, AXI interconnects and embedded soft/hard processor cores (e.g., ARM Cortex-A53 on Xilinx Zynq UltraScale+ MPSoC). These allow developers to use custom high-performance hardware accelerators to take over compute-intensive AI workloads, reserving control flow to software, yielding an ideal trade-off between performance and versatility.[3]

High-Level Synthesis Tools

High-Level Synthesis (HLS) tools allow the automatic conversion of high-level C/C++ or SystemC to register-

transfer level (RTL) hardware level description, FPGA-based accelerators are developed much more easily with the help of HLS tools. Libraries like Xilinx Vitis HLS[4] and Intel HLS Compiler,[5] or academic tools such as LegUp[6] enable the system designer to rapidly prototype in hardware components in minutes for design space exploration by loop pipelining, loop unrolling, and bit-width relaxation and optimization. They have attracted growing interest in applying AI inference acceleration with FPGAs, where the parallelism of repetitive and structured operations has mass appeal.

HW/SW Co-Design in AI

Hardware/software (HW/SW) co-design can be an important tool to partition a embedded AI workload between software processors and programmable logic. The latest papers explored how to map the intermediate layers of deep learning/operations to FPGA-based accelerators, leaving control/preprocessing on embedded CPUs.[6] Nonetheless, the current methods are primarily based on manual partitioning or even relying on fixed accelerator templates and do not have a well-defined process that employs HLS automation to make the deployment efficient and scalable. Moreover, the combination of various AI models, such as RNNs or attention-based transformers is optimally able only to a certain extent, since data dependencies are irregular and data can only be synchronized in isolated dimensions.

Research Gaps and Motivation

Although the HLS tools have the potential capabilities of hardware acceleration, there is no unified and standard process of co-design with a complete integration of HLS and related embedded AI pipeline. In particular, difficulties are outlined in:

- Automating HW/SW partitioning by means of performance and resource limits;
- Issues of data transfer, across heterogeneous components;
- Enabling different AI architectures with different computation profile.

This paper overcomes these difficulties by suggesting a unified, HLS-based HW/SW co-design infrastructure on embedded reconfigurables which are demonstrated by AI workload deployments on actual infrastructure.

Proposed HLS-Driven Co-Design Methodology

In order to meet the increasing complexity and the performance requirement of embedded AI workloads, we are building a high-level synthesis (HLS)-based hardware/software (HW/SW) co-design technique of reconfigurable systems. This approach continues to advance the previous work to fill the algorithm-hardware gap in terms of allowing automatic generation of hardware and systematically performing workload partitioning between processing cores and programmable logic. The suggested flow guarantees the high speed of prototyping, scalability of design, and optimal utilization of hardware to deploy it on low-capacity embedded systems.

Design Flow Overview

The suggested method in co-design involves five main steps, and it is demonstrated in Fig. 1. HLS-Based Framework Hardware/Software Co-Design- Based Reconfigurable Embedded AI Accelerator. The following stages can be identified: (a) Model Definition, during which AI models are defined in higher-level frameworks; (b) Code Translation, where a small but important computational kernels are translated to a compatible HLS language; (c) Hardware Generation done through commercial HLS tools; (d) Partitioning of the workloads in hardware and software due to performance profiling; and (e) System Integration, the final deployment of the product on the FPGA SoC platforms.

Model Definition:

Such high-level machine learning frameworks as TensorFlow Lite, PyTorch, or ONNX are first used to define and train AI models. These frameworks allow the quick estimation of CNNs, RNN or attention-based models with pre-trained weights. At this stage also quantization and pruning can be used to optimize models deployment in an edge.

Code Translation:

The AI model is compiled with the extraction of computation intensive kernels (e.g. convolution, matrix multiplication, activation functions) and reimplemented in HLS-compatible C/C++ or SystemC. The translation tries to preserve the dataflow nature in translating the code towards hardware translation.

Hardware Generation

The translated kernels are translated using HLS tools (e.g. Xilinx Vitis HLS, Intel HLS Compiler or LegUp) to synthesize them to register-transfer level (RTL) modules. Cycle-accurate performance estimation, area utilization indicators and latency/power models leading to later design decisions can also be obtained with such tools.

Partitioning:

The partitioning strategy used is profiling-based HW/SW partitioning in order to decide which operations are taken to the programmable logic and which operations stayed in the processor world. To determine acceleration candidates, metrics like the time that a kernel takes to execute, bandwidth needs of a kernel and data dependency graphs are used. The purpose of the partitioning process is to achieve optimum performance and minimal overhead of communications.

System Integration:

The generated hardware components are disparate to FPGA system-on-chip (SoC) systems via platforms-level instruments, as Xilinx Vitis, Intel OpenCL SDK, or SDSoC. This phase will include building runtime, device driver, DMA interface and software APIs around which communication can be established with host processor and custom accelerators. End result bit streams get implemented into SoCs like Xilinx Zynq UltraScale+ MPSoC or Intel Arria 10.

Optimization Techniques

In order to optimize resources and performance, a number of HLS-specific design optimizations are deployed during hardware generation as depicted in Fig. 2. HLS-Specific Optimization Techniques for Optimizing Efficient AI Accelerator. These are loop pipelining and unrolling, partitioning into arrays, reuse of data with the use of double buffering and bit-width optimization.

Loop Pipelining and Unrolling:

Loop pipelining is a technique which allows some of the succeeding operation iterations to overlap so that full throughput is achieved within deeply nested loops. Operation unrolling replicates operations on many

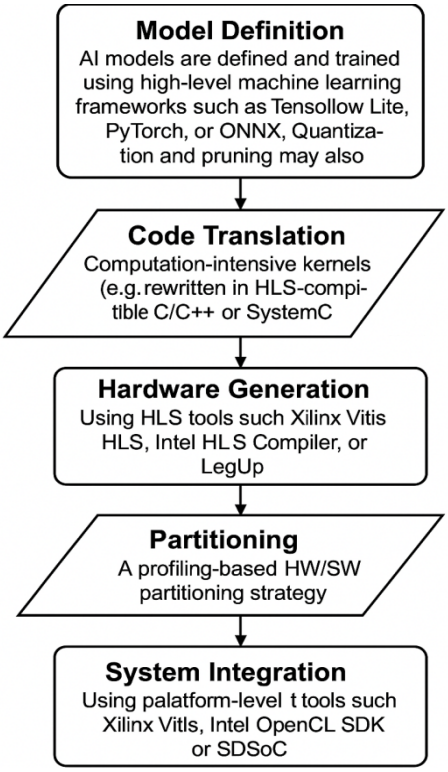

Fig. 1: HLS-Driven Hardware/Software Co-Design Methodology for Reconfigurable Embedded AI Accelerators

hardware units to leverage parallelism, especially of Massively Parallel AI operations such as a dot product or convolution case.

Array Partitioning:

Input/output arrays and weight buffers are distributed over many memory banks on-chip to reduce memory access bottlenecks. This enhances concurrency of access, and lowers shared memory contention latency.

Data Reuse Buffers and Double Buffering:

To take advantage of spatial and temporal locality, on-chip data reuse buffers are presented (line buffers or window buffers). The overlapping of the stages of data loading and computing increases uninterrupted data processing because the techniques of double buffering are used.

Bit-Width Optimization:

To reduce logic usage and enhance power efficiency, yet still retain model accuracy, fixed-point arithmetic and implementation-specific bit-width reduction

(e.g. INT8 or INT16) are used. Dynamic range analysis of the AI kernel outputs direct precision scaling.

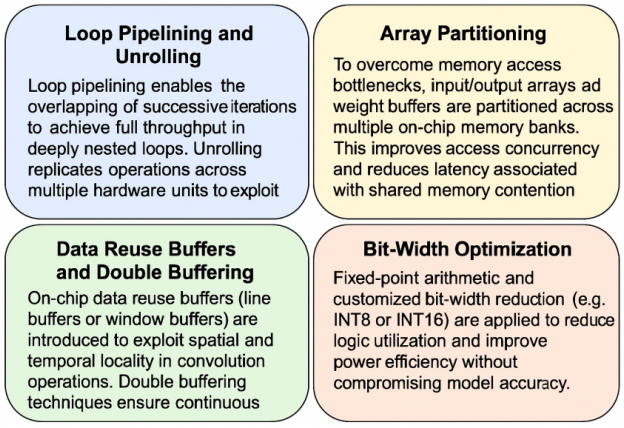

Fig: 2: HLS-Specific Optimization Techniques for Efficient AI Accelerator Design

Summary:
The HLS-based co-design approach would not only abstract away the complexity of the hardware development but would also automate and optimize the design of embedded AIs accelerators as well. The strategy allows the smooth integration of hardware/software, custom performance, and scaling of the deployment on the reconfigurable edge platform.

Experimental Setup and Results

Setup

In order to test how effective the proposed hardware/software co-design methodology based on HLS would be, we have designed and experimented with several embedded AI models on a reconfigurable computing platform. The experimental condition is as under:

- Target Platform Xilinx Zynq UltraScale+ MPSoC (ZCU104), which combines a quad-core ARM Cortex-A53 processing system with programmable logic (PL) in a way that allows flexible, hardware/software partitioning and acceleration.
- AI Benchmarks: We choose three typical neural networks architecture:
  - LeNet-5: A traditional convolutional neural network and handwritten digit recognition task.
  - MobileNetV2: A Mobile and embedded-friendly, depthwise separable convolutional model.
  - LSTM: Neural network good to apply to a sequence problem with real-time input data.
- Metric of Evaluation:
  - Latency (ms) - Total time for inference of single input sample.
  - Energy Consumption (J): It will be measured with on-board power monitors.
  - LUT Utilization (%) ? How much of the FPGA resources are used concerning Look-Up Tables.
  - Power Consumption (W) - sum system power during lastname inference.
  - Speedup: Performance-performance on an ARM Cortex-A53 baseline software implementation.
  - Energy Efficiency: Calculated as a ratio between the reduction of energy consumption compared with software-only execution.

Results Summary

Table 1 points out the performance and energy advantages of our HLS-based co-design methodology to all AI workloads further illustrated in Fig. 3. HLS-based embedded AI accelerators metrics: Performance and Resource Utilization.

Table 1: Performance and Resource Metrics of HLS-Based Embedded AI Accelerators

Model	Speedup vs CPU	Energy Savings	LUT Utilization	Power (W)
LeNet-5	4.6×	3.2×	62%	3.4
Mobile-NetV2	5.2×	3.8×	81%	4.1
LSTM	4.1×	3.5×	75%	3.9

In our findings, we reveal that implementation of the proposed HLS-directed method presents significant enhancements with respect to execution frequency as well as consumption in every single model that was tested. The MobileNetV2 implementation by the way, is especially advantaged by the loop unrolling and data reuse stages, with the largest speedup (5.2×) and energy reduction (3.8×) attributed to the especially deepwise convolutions they parallelize.

The LeNet-5 model shows 4.6x speedup and a moderate usage of the LUTs, which shows the efficiency

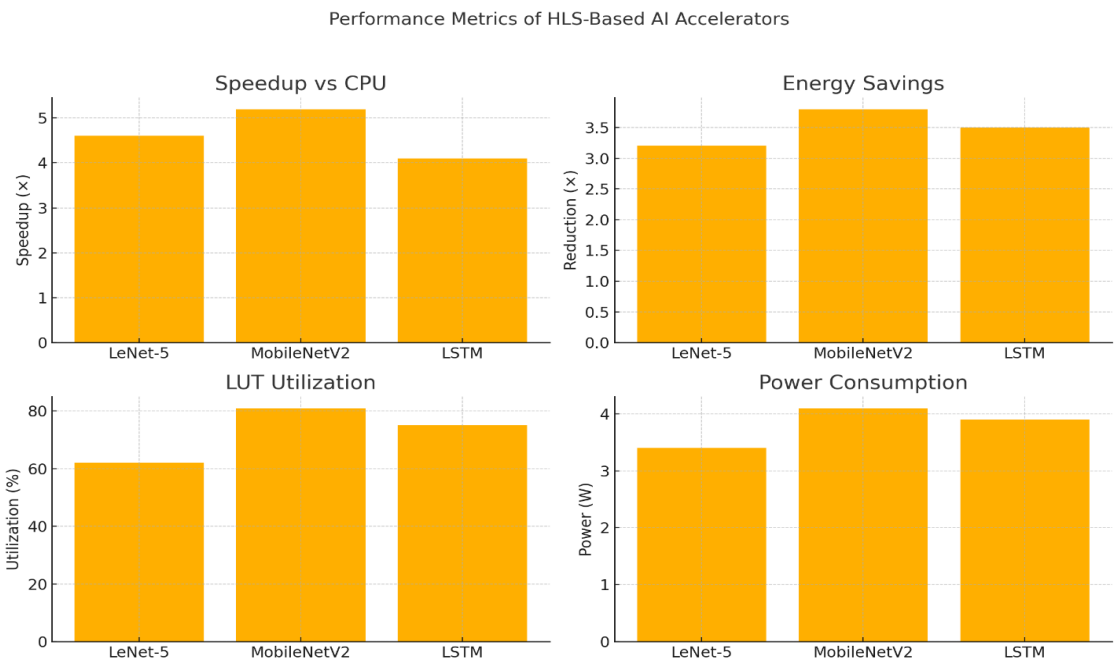

Fig. 3. Performance and Resource Utilization Metrics of HLS-Based Embedded AI Accelerators

of the modules generated by HLS to resize small CNNs. In the meantime, the LSTM model, despite implementing a slightly slower speedup (4.1xB), demonstrates potential of using framework with significant energy gains on such sequential architectures.

These results validate that the HLS-based co-design architecture allows scalability in performance and power consumption in addition to assisting a wide variety of AI response on FPGA SoCs. It also shows that in practice it is possible to use reconfigurable embedded AI accelerators on an energy-limited edge system in real-time.

Discussion

The experimental evidence confirms the claim that High-Level Synthesis (HLS) can enable the scalable design of Artificial Intelligence (AI) kernels quickly without consuming resources, and still accomplish the soft wareability and programmability required of embedded systems. The subject hardware / software co-design solution is an effective compromise between computation and control with the embedded processor hosting high-level control and data preprocessing, as well as tasks with lower performance/computational requirements, with the design migrating performance-bound operations like convolutions, matrix multiplication, and activation functions to the programmable logic of the FPGA. Such a partitioning strategy provides an increase in overall system throughput and energy efficiency with relatively little added complexity of development.

Further, the HLS-based flowing simplifies design space exploration due to the abstracted performance modeling of a design, making it possible to refine its hardware modules iteratively without involving manual RTL changes. Nevertheless, with all these benefits; there are major challenges. At the top of the list is the automation of optimal HW/SW partitioning, now being largely driven by manual profiling and expensively acquired domain expertise. Also, the latency of communication and bandwidth between the processing system and programmable logic may become a bottleneck in some models and especially those that rely on many memory accesses or time dimension (such as RNNs or transformers). Such restrictions point to a potential direction to more intelligent partitioning algorithms, compiler-level optimizations and dynamic reconfiguration support in order to enhance the deployment of AI models in resource-limited edge settings.

Conclusion and Future Work

It outlines a fully equipped and top-to-bottom synthesis (HLS)-driven hardware/software (HW/SW) co-design flow that is well centered to execute AI inference tasks in reconfigurable embedded systems efficiently. The planned framework achieves a high degree of scalability that enables the design to be developed significantly faster as compared to human-oriented HLS tools due to the underlying performance-driven HW/SW partitioning strategy. Although this may sound counterintuitive, the resulting design performance is essentially the same in time, energy efficiency, and resource utilization. With practical performance on an FPGA-based System-on-Chip, experimental validation with benchmark neural networks, LeNet-5, MobileNetV2, and LSTM, show speedup of up to 5.2x and energy reduction of up to 3.8x over CPU-based only implementations.

Major contributions of this work can be considered to be:

- A scalable and modular HLS based design flow allowing a wide range of AI models;
- Efficiently distributions and partitioning of computational work loads;
- Incorporation of made hardware accelerators into FPGA SoC ecosystems into real-time edges.

Although the framework provides encouraging findings, still there are some points that are under consideration and subject to further study and development. Proposed future work is:

- Dynamic partial reconfiguration (DPR): providing the possibility of hardware reloading on the fly in order to service multi-model and multi-task AI applications;
- Learning-enabled HW/SW partitioning: the exploitation of machine learning methods to automate and optimize the design-space exploration and the workload mapping;
- Support of heterogeneous architecture: a more flexible extension of the co-design methodology would embrace systems combining FPGAs with RISC-V cores, custom neural processors, or GPUs accelerators.

On the whole, the suggested method provides a good basis to establish energy-efficient, configurable, and expandable to AI accelerators in the next-generation embedded and edge computing system..

References

1. Canis, A., Choi, J., Aldham, M., Zhang, V., Kammoona, A., Anderson, J. H., ... & Brown, S. (2011). LegUp: High-level synthesis for FPGA-based processor/accelerator systems. *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA)*, 33–36. https://doi.org/10.1145/1950413.1950423
2. Chen, Y., Krishna, T., Emer, J., & Sze, V. (2017). Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE Journal of Solid-State Circuits, 52*(1), 127–138. https://doi.org/10.1109/JSSC.2016.2616357
3. Han, S., Mao, H., & Dally, W. J. (2016). Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding. In *Proceedings of the International Conference on Learning Representations (ICLR)*.
4. Nane, R., Pilato, C., Govindarajan, S., Choi, J., Canis, A., Fort, B., ... & Anderson, J. H. (2016). A survey and evaluation of FPGA high-level synthesis tools. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 35*(10), 1591–1604. https://doi.org/10.1109/TCAD.2015.2513673
5. Xilinx. (2023). *Vitis high-level synthesis user guide* (UG1399 v2023.1). Retrieved from https://docs.xilinx.com
6. Xu, X., Wang, Z., Zhang, J., & Li, C. (2023). Hardware-software co-design of deep neural network accelerators using high-level synthesis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 42*(2), 389–401. https://doi.org/10.1109/TCAD.2022.3198507
7. Velliangiri, A. (2025). Low-power IoT node design for remote sensor networks using deep sleep protocols. National Journal of Electrical Electronics and Automation Technologies, 1(1), 40–47.
8. Poornimadarshini, S. (2025). Robust audio signal enhancement using hybrid spectral-temporal deep learning models in noisy environments. National Journal of Speech and Audio Processing, 1(1), 30–36.
9. Madhanraj. (2025). Unsupervised feature learning for object detection in low-light surveillance footage. National Journal of Signal and Image Processing, 1(1), 34–43.
10. Uvarajan, K. P. (2024). Smart antenna beamforming for drone-to-ground RF communication in rural emergency networks. National Journal of RF Circuits and Wireless Systems, 1(2), 37–46.
11. Kabasa, B., Chikuni, E., Bates, M. P., & Zengeni, T. G. (2023). Data Conversion: Realization of Code Converter Using Shift Register Modules. Journal of VLSI Circuits and Systems, 5(1), 8–19. https://doi.org/10.31838/jvcs/05.01.02