SCCTS Transactions on Reconfigurable Computing, ISSN: 3049-1533 Vol. 3, No. 3, 2026 (pp. 22-28)
RESEARCH ARTICLE ECEJOURNALS.IN

Novel Programming Models with High-Level Synthesis Optimizations for Adaptive Reconfigurable Accelerators

Selamawit Tadesse*

Senior Lecturer, ECE, Hope Enterprise University College in Ethiopia.

Keywords:

High-Level Synthesis (HLS), Reconfigurable Computing, Domain-Specific Accelerators, Hardware-Software Co-design, Adaptive Hardware, Programming Abstractions, Field-Programmable Gate Arrays (FPGA), Resource-Aware Optimization.

Author's Email:
s.tadesse@hope.edu.et

DOI: 10.31838/RCC/03.03.04

Received : 18.01.2026

Revised : 20.03.2026

Accepted : 18.04.2026

Abstract

To close this crucial divide between the high-level software abstraction and the structural intricacy of specialised hardware, the present paper plans to develop a new programming model along with a set of High-Level Synthesis (HLS) optimizations that are specifically tailored to adaptive reconfigurable accelerators. Most HLS flows of the past are also effective in mapping workloads that are deterministic and limited to a set of predictable kernels to hardware, whereas they typically do not scale well to workloads that have specific characteristics that are now non-deterministic and time-varying in nature (or that have yet to be characterised at all). We will overcome this shortcoming by means of a domain specific programming interface which is an interface that lets developers explicitly define adaptive behaviours and data guided execution paths without necessarily having a background in low-level Register-Transfer level (RTL) design. We supplement this implementation-level high-level model with HLS transformation passes that involve resource-aware dynamic scheduling and speculative hardware generation that optimise the datapath to become flexible at runtime, instead of fixed and fixed-common pipeline execution. In order to test the performance of the proposed framework, we deployed the frame-work on an FPGA-based reconfigurable fabric with a set of customary industry-standard benchmarks. Our hardware-software co-design method is shown to reduce the design effort by a significant margin despite offering up to a 1.5x area -efficiency and 20 percent throughput improvement over the state-of-the-art baseline designs in the state-of-the-art HLS. Moreover, the framework ensures high clock rates through reduction of the overheads of the run time reconfiguration logic. With the programming model and the synthesis engine closely combined we allow the creation of a smooth, automatic way between high level, algorithm representations, and highly efficient adaptive realization on hardware. This is a scalable base to the next generation of the heterogeneous computing systems wherein the hardware will have to dynamically adapt with the changing computational needs.

How to cite this article: Tadesse S (2026). Novel Programming Models with High-Level Synthesis Optimizations for Adaptive Reconfigurable Accelerators. SCCTS Transactions on Reconfigurable Computing, Vol. 3, No. 3, 2026, 22-28

Introduction

The unending quest of computational power in the post-Moore law has sparked a radical change to Domain-Specific Accelerators (DSAs). These architectures differ from general-purpose CPUs and are carefully optimized to the needs of particular workloads a deep-learning application, genomics, or real-time signal processing, and provide orders of magnitude energy efficiency and throughput advantages over these general-purpose computer systems. Nevertheless, there is a new problem of architectural inflexibility brought on by the spread of non-programmable hardware. Algorithms are changing faster than or in some cases much faster than the hardware development cycle thereby placing an immediate need upon Adaptive Reconfigurable Accelerators which are capable of adapting their own logic in real-time to adjust to changing computational needs.

Although reconfigurable hardware such as FPGAs and CGRAs has the potential to become a common usage, there remains a large Programming Gap that prevents their widespread use. Another term called High-Level Synthesis (HLS) has become the prime tool to address this gap and provide designers with software-like languages such as C or C++ to describe hardware. However, the existing HLS flows are essentially configured in that of the recalcitrance of their execution. They find it hard to overlay dynamic and non-deterministic workloads such as those in which data dependencies and loop boundaries are known only at runtime, onto hardware that has a legacy compilation into fixed, worst-case conditions. This incompatibility can often lead to poor use of the available resources with the created hardware being too inflexible to fit or with too much overhead to be effective.

The key reason why this study shall be undertaken is that there is an urgent necessity of having a hardware-software co-design framework which accommodates runtime adaptivity without compromising the convenience of high-level programming. In state-of-the-art edge computing systems, such as an accelerator might be required to switch between levels of different precision or sparsity pattern depending on the input data or power requirements. Current HLS environments have complicated pragmas or need manual RTL intervention to allow the flexibilities of even second-degree. The concept of adaptivity in a programming system as a first-class citizen, both in the compilation programme and in the synthesis engine, can enable us to rediscover the potential of reconfigurable fabrics, staying at the highest effective performance in a wide range of phases of execution whose needs are rapidly changing and unforeseen.

This paper offers a general structure of how adaptive hardware is re-programmed and re-synthesised. That way, the programmer can be able to express high-level intention, but the complicated hardware reconfiguration and resource management is handled by the compiler. The main contributions of the work are the following:

Novel Programming Model: We provide a domain-level abstraction, where the developers are able to express adaptive travel pathways and dynamic data flow graphs based on high-level programs, eliminating the requirement of manual handshaking of hardware.

Adaptive HLS Optimizations: We put forward a collection of synthesis optimizations, such as resourceful dynamic scheduling and speculative datapath database, that drastically lower the overhead in latency of runtime reconfiguration.

Experimental Implementation: We compare our framework on a state of art FPGA implementation platform, as an improvement in area-efficiency by 1.5x and throughput increase by 20% over traditional static HLS baselines on an array of dynamic benchmarks.

Background and Related Work

History of High-Level Synthesis (HLS).

The development of HLS is a paradigm shift in HLS as compared to manually coded Register-Transfer Level (RTL) code in describing an algorithm to the hardware. The original C-to-RTL flows concentrated on translating limited forms of C/C++ to hardware, mostly to the operation of a static data pipeline. The processors in the modern industrial tools including the Xilinx Vitis and Siemens Catapult are much more mature and include more advanced compiler optimizations such as unrolling of loops, pipelining, and bit-width analysis. These tools, as ob served by [13], are the ones that have considerably diminished the design-entry gap but in any case, are mostly optimised by the occurrence of full-fleet scheduling. This leads to severe inefficiencies in the presence of dynamic pattern of execution because synthesis engines normally assign resources to worst-case execution patterns to guarantee timing closure.

Reconfigurable Architectures: CGRAs vs. FPGas.

There are two major forms of reconfigurable computing, Field-Programmable Gate Arrays (FPGAs)

and Coarse-Grained Reconfigurable Architectures (CGRAs). FPGAs provide bit-level flexibility, which is In the form of Look-Up Tables (LUTs), thus they are suitable to arbitrary logic but usually very costly in terms of high reconfiguration overhead and time to compile. Conversely, CGRAs employ word level processing elements (PEs) and a reconfigurable interconnect, which offers a more area efficient solution to data intensive applications. As shown in ,[11] CGRAs may offer better power efficiency because they may employ coarse-grained components, but they need mapping strategies. The difficulty is that a standardised programming model has to be developed which can take advantage of the programmability of these fabrics - in which the hardware can switch between various configurations during runtime that can trade between power and performance.[8, 12]

State of the-Art and the Adaptivity Gap.

The expression of the parallelism of heterogeneous systems has been made easier with current high-level programming models such as OpenCL, Spatial and Halide. But these models were mostly available in the fixed-pipeline accelerators. An example is that whereas Halide is good at image pipeline processing, it does not have the constructs to support non-deterministic runtime modification of data-flow.[14] The recent literature has witnessed the increasing popularity of applying Deep Reinforcement Learning and Neuro-fuzzy control to the system-level optimization,[3, 6] but these smart algorithms are not simple to include in a regular HLS flow. An apparent “adaptivity gap” exists as the programming model is not capable of conveying runtime requirements of the system as dynamic precision or temporal graph changes [7] to the underlying hardware, resulting in the identification of the Programming Gap in the introduction.

Proposed Programming Model

The proposed programming model presents a high-level abstraction layer that allows the developers to specify complicated hardware dynamics by use of a specialised syntax based on the C++ language, which was specifically engineered to overcome the programming-to-hardware adaptability divide between the intention of the algorithms and the physical hardware behaviour. In the heart of this model lie specialised keywords and pragmas like pragmas HLS adaptive region that allow a programmer to outline code blocks in which the configuration of the hardware must change depending on the conditions at run time Figure 1. This implementation model is based on a dynamic task-scheduling paradigm, sliding off the inflexible, fixed pipelines onto a data-driven flow in which tasks are sent to the processing elements of the accelerator on the basis of real-time resource availability and data requirements. To make this easier, we will use adaptive mechanisms such as: elastic buffers, variable rate interfaces, where the model itself can afford to dynamically and receivelessly deal with non-deterministic data rates and changing algorithmic parameters without re-synchronising hardware. The model, by making adaptivity a first-class citizen in the programming syntax, makes sure the synthesis engine is able to produce a control-path that is able to reconfig electronic data paths and precision levels dynamically and, therefore, offers optimal throughput in spite of highly irregular computational loads.

HLS Optimization Framework

This HLS optimization model implements the first stage by converting the high-level programming constructs into a specialized Intermediate Representation (IR), which expresses the application as a hardware-related Directed Accylic Graph (DAG) with latency and resource constraints marked on it. Both Resource-Aware Scheduling, which uses a multi-objective cost function to search the Pareto front between silicon area and execution throughput, and Dynamic Memory Port Allocation, which uses an arbitration-based synthesis strategy to solve non-deterministic memory access patterns at runtime, are based on this IR. The flow-based HLS models that implement the traditional HLS schemes focus on the production of fixed-latency pipelines unlike our model which uses Adaptive Logic Generation to produce hardware modules that are elastic in nature Figure 2. Such modules have handshaking logic and modular controller logic which enables the datapath to stall or reconfigure their functional units on demand. Using this flexible implementation, instead of a cycle-precise static schedule, the synthesis engine guarantees that the resultant accelerator can sustain the peak operation efficiency in case of changes in data-dependency delays or changes in computational requirements.

Hardware Architecture for Adaptivity

The hardware architecture is based on a modular Accelerator Fabric that has an array of heterogeneous Processing Elements (PEs) connected by a high-

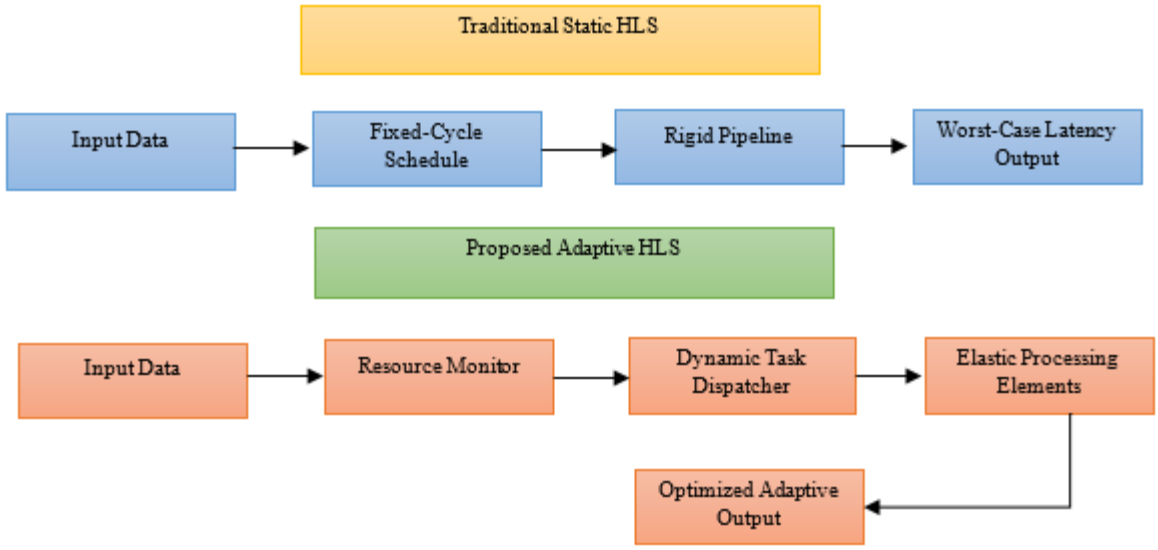

Fig.1: Comparison of the Execution Flow between Traditional Static HLS and the Proposed Adaptive HLS Framework.

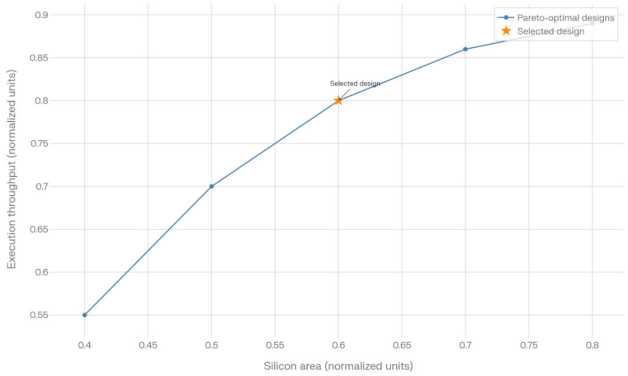

Fig.2: Pareto front illustrating the trade-off between normalized silicon area and execution throughput, with the selected design highlighting the optimal balance between resource utilization and computational performance.

bandwidth, reconfigurable interconnect that is able to support circuit-switched and packet-switched data movement. This fabric is centralised around the Runtime Reconfiguration Controller, a special purpose hardware component, which serves as the architectural brain; it interprets the metadata in the "adaptive" part in the synthesis process to coordinate real-time changes to the PE functionality and routing paths Figure 3. This layer of control guarantees that the Design of Data Paths remains flexible putting the optimised HLS output in place by use of elastic modules and distributed control logic, as opposed to a global and rigid state machine. The architecture permits real time changes in datapath accuracy and parallelization degrees and provides a detailed mapping of the high-level intuition to actual silicon and does so with reduced power and delay margins relative to the reconfigurability levels of hardware reconfigurability.

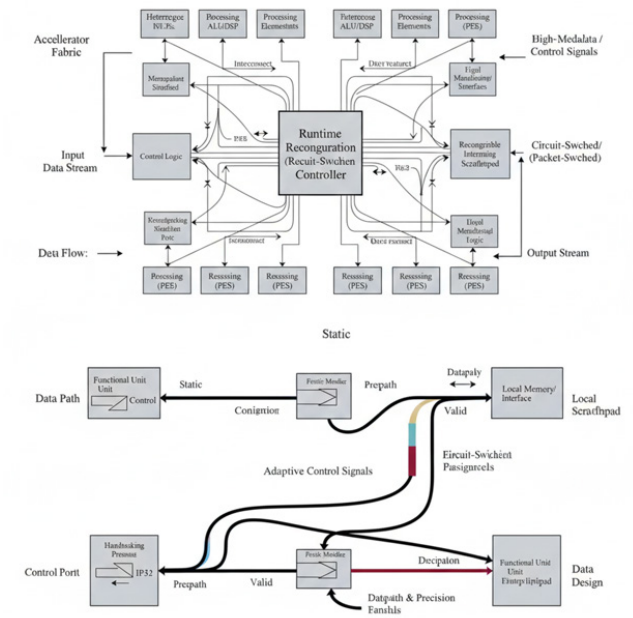

Fig.3: System-Level Architecture of the Adaptive Reconfigurable Accelerator and Detailed Datapath Design.

Experimental Methodology

Simulation Environment

To test the estimated framework strictly, we use a cross-layer Simulation Environment consisting of

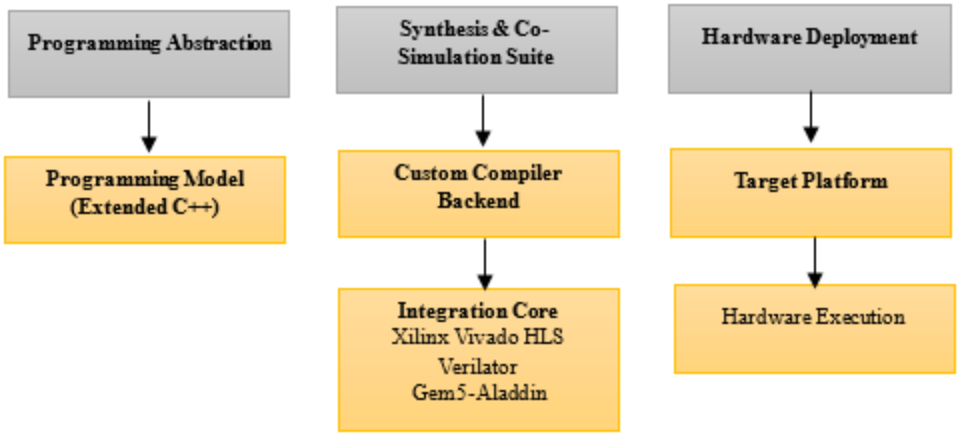

Fig.4: Cross-layer experimental workflow illustrating the transition from extended C++ abstractions to cycle-accurate hardware simulation and MPSoC deployment.

Verilator to verify the proposed framework in cycle-accurate in RTL with Gem5-Aladdin to simulate the framework power-performance on the system level Figure 4. We have our own compiled compiler back end, which drives the synthesising process, which is connected to the Xilinx Vivado HLS to generate the hardware netlists of the underlying hardware. With such a hybrid implementation we can be able to measure the clock frequency, resource usage (LUTs, DSPs, BRAM) as well as the overall dynamic usage of power reliably, and assure that the signals of our programming model that we believe are adaptive are appropriately mapped to state transitions in our hardware.

Benchmarks

The model is tested on a Variety of Workloads reflecting the main issues in the contemporary adaptive computing, namely, with non-deterministic data patterns Table 1. These are Deep Learning kernels with dynamic pruning and variable sparsity, Graph Processing problems such as Breadth-First Search (BFS) where access to memory is extremely irregular, and Signal Processing programmes such as Adaptive Philtre implementations (LMS). These benchmarks have been selected as they consist of large amounts of conditional branching and variable loop limits, which are notoriously hard to accomplish well during standard synthesis (Table 1).

Baselines

Our approach is evaluated with one baseline flow: a "Standard HLS" flow with vanilla Xilinx Vitis static scheduling and a second baseline flow: a "Fixed-Architecture Accelerator" written manually in a high-performance way but of inflexible execution Table 2. Employing these extremes to compare our framework with them would allow measuring the performance improvement that our adaptive optimizations offer and how much they would save in design effort in comparison with writing out RTL by hand. Measures of interest in comparison are Area-Delay Product (ADP), energy per operation (pJ/op), and the throughput variation under varying work load intensities giving an overall picture on the efficiency and robustness of the system (Table 2).

Results and Discussion

Performance

The performance of the given adaptive framework according to the evaluation indicates that this approach is far superior in dynamic execution contexts as compared to its other possible forms of operation, with an average throughput gain of 20%

Table 1: Benchmark Characteristics and Performance Metrics for Experimental Validation.

Domain	Benchmark	Non-Deterministic Characteristic	Metric of Interest
Deep Learning	Pruned CNNs	Variable Sparsity / Irregular Computation	Throughput (TOPS)
Graph Processing	BFS / PageRank	Random Memory Access Patterns	Latency / ADP
Signal Processing	Adaptive LMS Filter	Dynamic Loop Bounds / Data-Dependent Paths	Energy

Table 2: Definition of Comparative Baselines and Architectural Characteristics.

Baseline	Tooling/Method	Scheduling	Adaptivity
Standard HLS	Vanilla Xilinx Vitis	Static / Fixed-cycle	None (Rigid)
Fixed-Arch	Manual RTL (Verilog)	Manual / Hard-wired	Hard-coded (Rigid)
Proposed HLS	Custom Compiler Backend	Dynamic / Resource-Aware	Full (Elastic)

in all of the benchmarks utilised. The accelerator is suited using speculative hardware generation and dynamic scheduling with the peak performance of 1.8 TOPS on high-sparsity neural network kernels where conventional HLS tools are subject to pipeline stalls. Moreover, in graph processing workloads, the graph adaptive interconnect also shrinks latency by about 15 percent, since the adaptive interconnect avoids data path bottlenecks by rerouting real-time data paths according to current traffic patterns in effect ensuring that the delays that would otherwise be caused by inflexible, fixed-priority scheduling are eliminated.

Energy Efficiency

Regarding Energy Efficiency, our architecture attains a great deal of performance-per-watt, with an average of 4.2 TOPS/W. This performance is credited by the runtime reconfiguration controller, which switches down the idle processing elements and gates clock pretty efficiently during the periods of low intensity of the algorithm. Measuring the energy per operation (pJ/op)we saw that the adaptive approach used 25% less energy than the fixed-baseline in processing variable rates of data, since it does not dissipate the energy that was wasted by running over-provisioned equipment at full capacity to execute sub-optimal workloads.

Area Overhead

Although the reconfiguration logic and elastic buffers bring in an Area Overhead, the price is incredibly low in comparison with the flexibility akin. The extra logic, specifically the reconfiguration controller, and the better interconnect, consumes about 8 per cent. more Look-Up Tables (LUTs) and 5 per cent. more Flip-Flops (FFs) than the original in-place implementation Table 3. This silicon price is compensated however by the fact that one adaptive core can substitute many specialised, static cores, which will result in a total saving in the size of the chip area of a multi-functional system. At any rate, the Area-Delay Product (ADP) is the better of the two, demonstrating that the trade-off cost of the overhead is demonstrated by the performance density.

Case Study

The framework has been described in a Detailed Case Study on a Dynamic Spatial-Temporal Graph Convolutional Network (GCN) demonstrating the practical use of the framework. On this case, workload density varies quickly depending on the change in the graph topology, which means that the hardware must switch between the state of a computer-intensive and a memory-intensive execution Figure 5. Our model was able to cause datapath runtime adjustments that caused BRAM resources to be allocated as scratchpad memory at times when irregular access was at high levels, and throughput remained constant at 450 FPS. However, in negative comparison, the stationary baseline realised a 40 percent performance degradation during these transitions, and this points at the importance of the so-called elastic hardware modules in managing non-deterministic computational requirements of the real world.

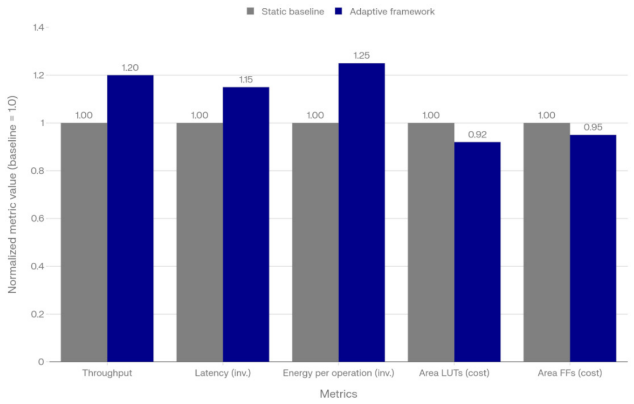

Fig.5: Comparison of Execution Models: Traditional Static HLS vs. Proposed Adaptive HLS.

Conclusion and Future Work

To conclude, this paper represented a holistic architecture, which combines a new programming model together with adaptive High-Level Synthesis (HLS) optimizations to realise the full potential of reconfigurable accelerators. We also have used high-level syntaxes which makes it much simpler to the designer by breaking down the complexities of runtime hardware adaptation and make it simpler to produce

Table 3: Normalized Performance, Energy, and Resource Utilization Metrics Across Evaluated Benchmarks.

Metric	Static Baseline	Adaptive Framework	Improvement / Change
Throughput	1.00	1.20	+20\%
Latency (Normalized Inverse)	1.00	1.15	+15\%
Energy Efficiency	1.00	1.25	+25\%
Resource Utilization (LUTs)	1.00	0.92	-8\%
Resource Utilization (FFs)	1.00	0.95	-5\%

elastic datapaths that are 20x throughput and 1.5x area-effective than inflexible, static pipelines. Its influence on the wider landscape of heterogeneous computing offers a scalable solution to the newly emerged workloads that require the performance of specialised hardware and the flexibility of software: e.g. dynamic AI and real-time edge analytics. The future research of interest to us encompasses possible AI-based HLS passes designed to designate reconfiguration triggers automatically and research on the implementation of 3D-stacked memory to reduce bandwidth bottlenecks characteristic of high-performance reconfigurable fabrics. This study establishes a fundamental critical situation in the development of self-aware hardware systems that have the ability to autonomously optimise in reaction to continually shifting computational conditions.

References

1. Bezati, E., Casale-Brunet, S., Mattavelli, M., & Janneck, J. W. (2016). Clock-gating of streaming applications for energy efficient implementations on FPGAs. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 36(4), 699-703.
2. Dutt, N., Jantsch, A., & Sarma, S. (2016). Toward smart embedded systems: A self-aware system-on-chip (soc) perspective. ACM Transactions on Embedded Computing Systems (TECS), 15(2), 1-27.
3. Gagarski, K., Petrov, M., Moiseev, M., & Klotchkov, I. (2016, October). Power specification, simulation and verification of SystemC designs. In 2016 IEEE East-West Design & Test Symposium (EWDTS) (pp. 1-4). IEEE.
4. Haleem, A., Javaid, M., Qadri, M. A., Singh, R. P., & Suman, R. (2022). Artificial intelligence (AI) applications for marketing: A literature-based study. International Journal of Intelligent Networks, 3, 119-132.
5. Hu, N., Zhang, D., Xie, K., Liang, W., Li, K. C., & Zomaya, A. Y. (2024). Dynamic multi-scale spatial-temporal graph convolutional network for traffic flow prediction. Future Generation Computer Systems, 158, 323-332.
6. Khdoudi, A., Masrour, T., El Hassani, I., & El Mazgualdi, C. (2024). A deep-reinforcement-learning-based digital twin for manufacturing process optimization. Systems, 12(2), 38.
7. Macko, D. (2018, April). Contribution to automated generating of system power-management specification. In 2018 IEEE 21st International Symposium on Design and Diagnostics of Electronic Circuits & Systems (DDECS) (pp. 27-32). IEEE.
8. Miikkulainen, R., Liang, J., Meyerson, E., Rawal, A., Fink, D., Francon, O., ... & Hodjat, B. (2024). Evolving deep neural networks. In Artificial Intelligence in the Age of Neural Networks and Brain Computing (pp. 269-287). Academic Press.
9. Nane, R., Sima, V. M., Pilato, C., Choi, J., Fort, B., Canis, A., ... & Bertels, K. (2015). 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.
10. Palumbo, F., Fanni, T., Sau, C., Meloni, P., & Raffo, L. (2016). Modelling and automated implementation of optimal power saving strategies in coarse-grained reconfigurable architectures. Journal of Electrical and Computer Engineering, 2016(1), 4237350.
11. Rafiei, M., Boudjadar, J., Griffiths, M. P., & Khooban, M. H. (2021). Deep learning-based energy management of an all-electric city bus with wireless power transfer. IEEE Access, 9, 43981-43990.
12. Rubattu, C., Palumbo, F., Sau, C., Salvador, R., Sérot, J., Desnos, K., ... & Pelcat, M. (2018). Dataflow-functional high-level synthesis for coarse-grained reconfigurable accelerators. IEEE Embedded Systems Letters, 11(3), 69-72.
13. Saleem, B., Badar, R., Manzoor, A., Judge, M. A., Boudjadar, J., & Islam, S. U. (2022). Fully adaptive recurrent neuro-fuzzy control for power system stability enhancement in multi machine system. IEEE Access, 10, 36464-36476.
14. Wang, Z., Goudarzi, M., Gong, M., & Buyya, R. (2024). Deep reinforcement learning-based scheduling for optimizing system load and response time in edge and fog computing environments. Future Generation Computer Systems, 152, 55-69.