Comparative Analysis of Programming Models for Reconfigurable Hardware Systems

João Carlos da Silva[1], Maria Luísa de Oliveira Souza[2], Antônio de Almeida[3]*
[1,2]Department of Electrical Engineering, Londrina State University (UEL), Londrina 86057-970, Brazil

Keywords:
Hardware Abstraction;
Programming Models;
Reconfigurable Hardware;
System Optimization;
Performance Evaluation;
Software-Hardware Co-design

Corresponding Author Email:
almeidadeant@uel.br

DOI: 10.31838/RCC/02.01.02

Received : 12.07.24
Revised : 07.10.24
Accepted : 05.12.24

Abstract

Field programmable gate array (FPGA) based reconfigurable computing systems are shown to have great potential for accelerating computationally intensive applications. To date, however, these systems have had to be programmed with specialized hardware design skills, making them less accessible. These models and tools, which aim at simplifying FPGA development, are examined in this article, and the ease of use, performance, and generational efficiency in producing hardware designs are compared among them. This has allowed the use of reconfigurable hardware through high level synthesis (HLS) tools without having in depth hardware design knowledge. We will demonstrate imperative, functional, and graphical programming paradigms via Impulse C, Mitrion-C, and DSPLogic. Through analysis of the programming models, development workflows and results obtained across multiple benchmark applications, we can discern the tradeoffs between performance and productivity for reconfigurable computing.

How to cite this article: Silva JC, Souza MLO, de Almeida A (2025). Comparative Analysis of Programming Models for Reconfigurable Hardware Systems. SCCTS Transactions on Reconfigurable Computing, Vol. 2, No. 1, 2025, 10-15

Reconfigurable Systems: Programming Models

The purpose of programming model is to define what the hardware abstraction looks like to the developers and which architectural details we are going to expose and how data transfers and computations will be expressed. Let's examine the key characteristics of different programming paradigms for FPGAs.

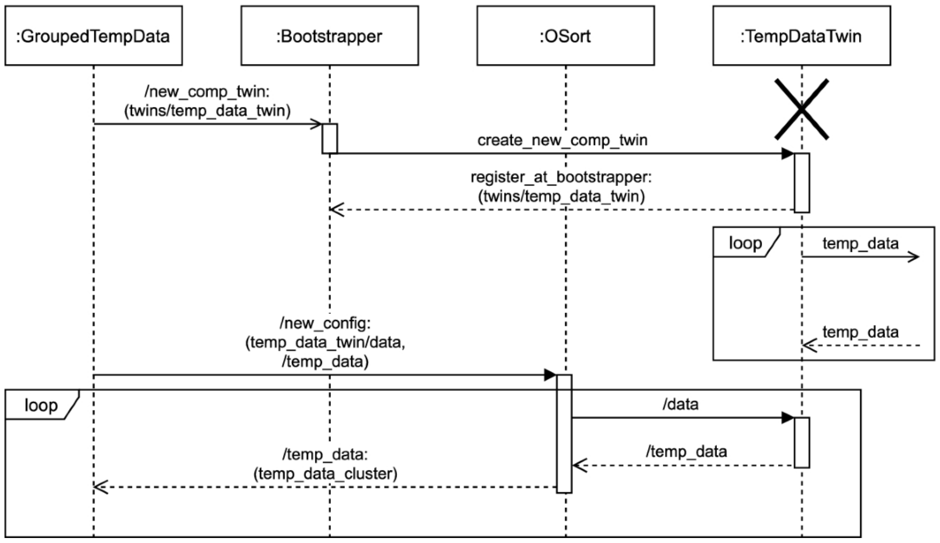

Fig. 1: Reconfigurable Systems: Programming Models.

C based HLS

For parallelism and interprocess communication, the standard C or its syntax extensions, such as Impulse C, are extended with tools. Sequential programming model, based on Familiar C syntax. Command matrix pragmas, and library functions for guiding hardware generation. Automatic instruction level parallelism. Processes and streams for support of task and pipeline parallelism. Integration with existing HDL IP coresels for Reconfigurable Hardware Systems. Reconfigurable computing systems that leverage field-programmable gate arrays (FPGAs) offer immense potential for accelerating computationally intensive applications. However, programming these systems has traditionally required specialized hardware design skills, limiting their accessibility. This article examines various high-level programming models and tools aimed at simplifying FPGA development, comparing their ease of use, performance, and efficiency in generating optimized hardware designs (Table 1).

Programming Models for Reconfigurable Systems

- Familiar C-like syntax and sequential programming model
- Pragmas and library functions to guide hardware generation
- Automatic extraction of instruction-level parallelism
- Support for task-level and pipeline parallelism through processes and streams
- Integration with existing HDL IP cores

Table 1: Programming Models for Reconfigurable Hardware Systems

Model	Feature Overview
Hardware-Software Co-Design	Hardware-software co-design allows for the joint development of hardware and software components, optimizing system performance and resource utilization in reconfigurable hardware.
High-Level Synthesis	High-level synthesis provides an abstraction that simplifies the design process, enabling the automatic conversion of high-level algorithms to hardware implementations.
Dataflow Programming	Dataflow programming models are ideal for reconfigurable hardware as they map tasks to hardware components based on the flow of data, enabling efficient parallelism.[1-5]
Implicit Parallelism	Implicit parallelism involves identifying and exploiting parallel operations in programs without the need for explicit parallel constructs, improving execution speed in reconfigurable hardware.
Reconfigurable Computing Languages	Reconfigurable computing languages provide domain-specific constructs for designing and simulating hardware that can be reconfigured during runtime, offering flexibility and performance.
Virtualized Programming Models	Virtualized programming models abstract the underlying hardware from the software, allowing for resource sharing and efficient execution of multiple tasks on reconfigurable systems.

Table 2: Performance Characteristics of Programming Models for Reconfigurable Systems

Characteristic	Measurement Criteria
Performance Efficiency	Performance efficiency measures how well a programming model translates algorithms into hardware operations, optimizing both speed and computational power [6]-[9].
Hardware Utilization	Hardware utilization evaluates how effectively the reconfigurable hardware is used, ensuring that available resources are maximally employed during computations.
Scalability	Scalability assesses the ability of the programming model to handle increasing complexity and larger datasets without significant performance degradation.
Resource Flexibility	Resource flexibility indicates the model, capacity to adapt and reallocate hardware resources based on changing system demands or application needs .[10-15]
Development Complexity	Development complexity considers the ease of using the programming model for system designers, with a simpler model reducing the learning curve and development time.
Energy Efficiency	Energy efficiency measures the model‚Äôs ability to minimize power consumption while achieving the required computational performance, which is critical in embedded or mobile systems.

This cuts the learning curve to software developers at the cost of handshaking control over generated hardware (Table 2).[16-18]

- Familiar C-like syntax and sequential programming model
- Pragmas and library functions to guide hardware generation
- Automatic extraction of instruction-level parallelism
- Support for task-level and pipeline parallelism through processes and streams
- Integration with existing HDL IP cores

This approach offers a gentler learning curve for software developers but may limit fine-grained control over generated hardware.

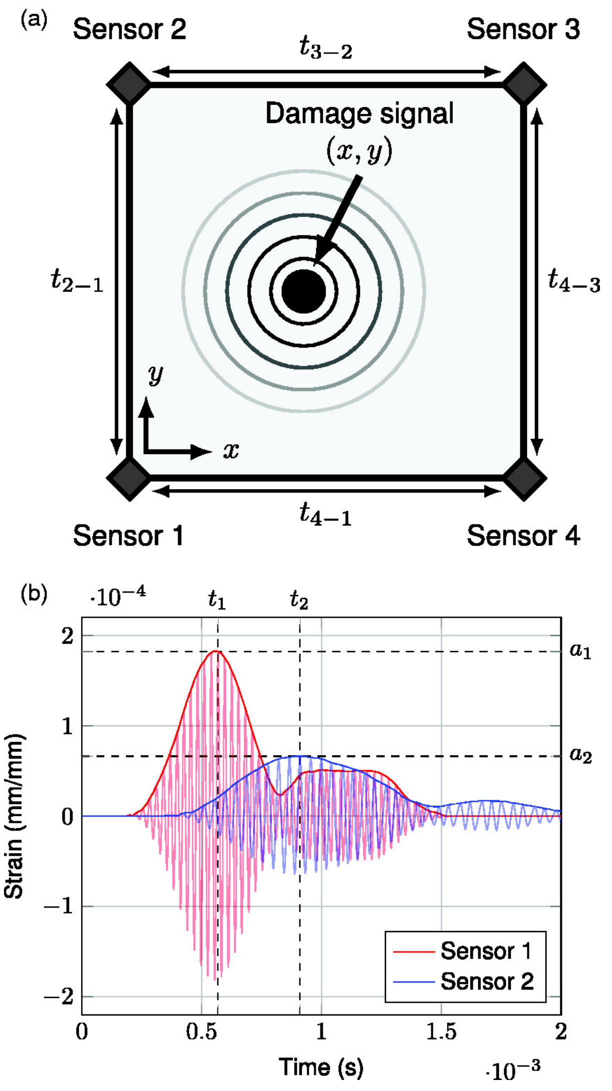

Fig. 2: Functional Programming for FPGAs

Functional Programming for FPGAs

Functional languages like Mitrion-C take a radically different approach:

Mitrion Virtual Processor

Unlike other modern HLS tools, Mitrion-C transcends generating RTL to output a configuration file for the Mitrion Virtual Processor (MVP). The target FPGA is serviced by a massively parallel soft core, optimized for the MVP.[19-20]

Development Process

1. Mitrion C implementation of an algorithm
2. Mitrion SDK instrumentation and compilation and simulation.
3. MVP configuration generation
4. Host application integration

Generation of MVP for the target FPGA. Very expressive for different classes of algorithms. Exploitation of fine-grained parallelism automatically. Deterministic execution model. Steep learning curbs for imperative progamers. No compatibility with existing HDL IP. Overhead potential to MVP architectureional HLS tools that generate RTL directly, Mitrion-C compiles to a configuration for the Mitrion Virtual Processor (MVP). The MVP is a massively parallel soft core optimized for the target FPGA.[21-23]

1. Algorithm implementation in Mitrion-C
2. Compilation and simulation with Mitrion SDK
3. MVP configuration generation
4. Integration with host application
5. Synthesis of MVP for target FPGA (Figure 3)

Advantages and Challenges

Advantages:

- Highly expressive for certain algorithm classes
- Automatic exploitation of fine-grained parallelism
- Deterministic execution model

Challenges:

- Steep learning curve for imperative programmers
- Limited compatibility with existing HDL IP
- Potential overhead of MVP architecture

DSPLogic: Signal Processing with Graphical

We use DSPLogic as a visual approach to FPGAs programming, particularly for digital signal processing

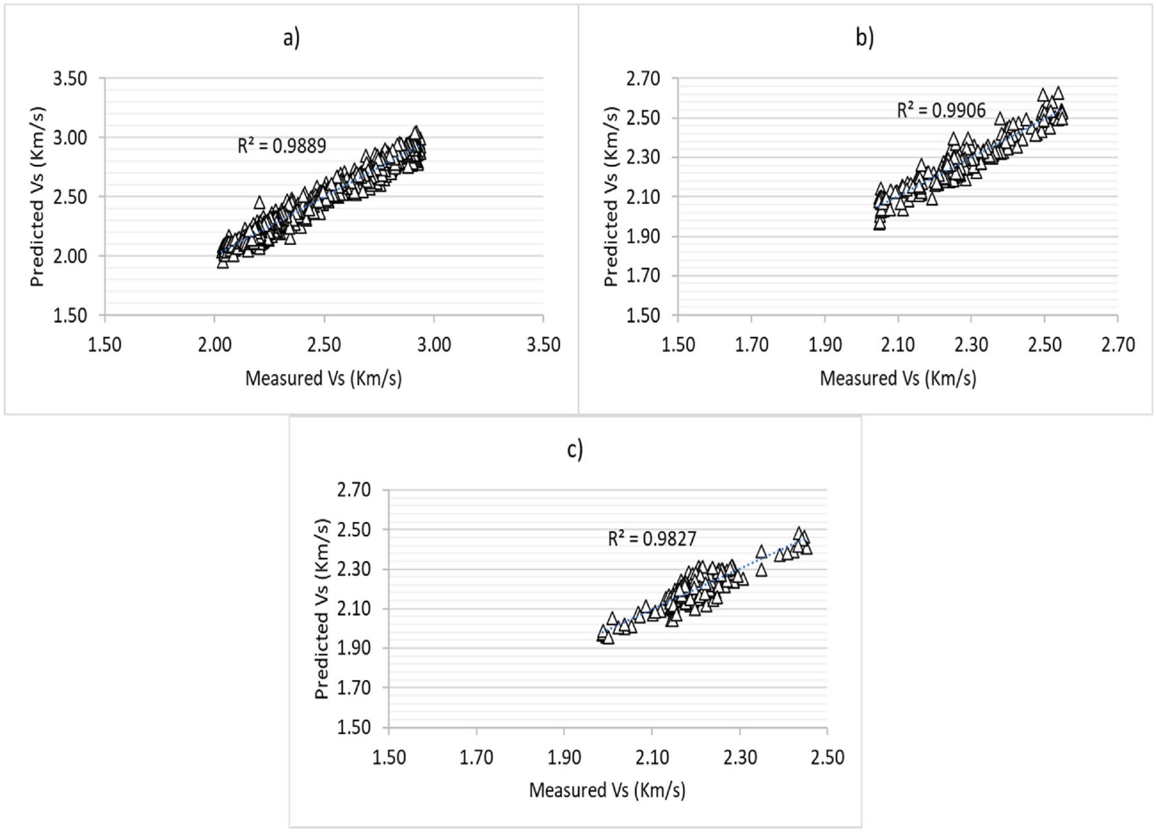

Fig. 3: Development Process

applications. FPGA Implementation custom block library. Support for Xilinx System Generator. Block diagram to HDL automatic generationHLS tools that generate RTL directly, Mitrion-C compiles to a configuration for the Mitrion Virtual Processor (MVP). The MVP is a massively parallel soft core optimized for the target FPGA. DSPLogic represents a visual approach to FPGA programming, particularly suited for digital signal processing applications. Let's examine its unique features: DSPLogic leverages Simulink's graphical environment, extending it with. Hardware in the Loop Testing and Verification. Also intutitive for DSP algorithm designers. Rapid prototyping, and design space exploration. Seamless integration with MATLAB for algorithm development. Less flexible in general purpose computing. May lead to less efficient hardware in non DSP applications. Simulink environment familiarity is requiredS tools that generate RTL directly, Mitrion-C compiles to a configuration for the Mitrion Virtual Processor (MVP). The MVP is a massively parallel soft core optimized for the target FPGA.[10-14] DSPLogic represents a visual approach to FPGA programming, particularly suited for digital signal processing applications. Let's examine its unique features.[24]

Programming Models comparison

Having examined the key characteristics of each programming model, let's compare their effectiveness across our benchmark applications. Ease of use and learning curve. C programmers have a moderate learning curve. Some hardware thinking is needed for explicit parallelisms that generate RTL directly, Mitrion-C compiles to a configuration for the Mitrion Virtual Processor (MVP). The MVP is a massively parallel soft core optimized for the target FPGA.

Conclusion

In particular, high level programming models for reconfigurable computing bring productivity gains in the order of magnitude by providing software developers increased flexibility to use FPGA acceleration, but without the need for significant hardware design expertise. While these tools come with tradeoffs between ease of use, performance and efficiency. Impulse C embodies imperative approaches similar to Impulse C, which give software developers a familiar point of entry, but might come at some reduction in performance. Mitrion-C provides powerful abstractions for some algorithm classes, but at a steeper learning curve than functional models. Specific domains tend to be good targets for graphical tools like DSPLogic, but my impression is that such tools tend to be less flexible when used for general purpose computation. Application requirements, developer expertise and platform are the main drivers when choosing which programming model. As these tools mature, they provide the ability to transform reconfigurable computing from a technology of academic interest to one of accessible application acceleration capability by diverse developers.

References:

1. Öksüz, E., Altun, A., & Özen, A. (2016, May). A frequency domain channel equalizer for discrete Wavelet Transform based OFDM systems. In *2016 24th Signal Processing and Communication Application Conference (SIU)* (pp. 1153-1156). IEEE.
2. Alves, T. M., & Cartaxo, A. V. (2015). Virtual carrier-assisted direct-detection MB-OFDM next-generation ultra-dense metro networks limited by laser phase noise. *Journal of Lightwave Technology*, *33*(19), 4093-4100.
3. Kumar, V., Mukherjee, M., Lloret, J., Ren, Z., & Kumari, M. (2021). A joint filter and spectrum shifting architecture for low complexity flexible UFMC in 5G. *IEEE Transactions on Wireless Communications*, *20*(10), 6706-6714.
4. Murata, T., Ishibuchi, H., & Tanaka, H. (1996). Multi-objective genetic algorithm and its applications to flowshop scheduling. *Computers & industrial engineering*, *30*(4), 957-968.
5. Musharavati, F., & Hamouda, A. S. M. (2012). Enhanced simulated-annealing-based algorithms and their applications to process planning in reconfigurable manufacturing systems. *Advances in Engineering Software*, *45*(1), 80-90.
6. Nourelfath, M., Ait-Kadi, D., & Soro, I. (2002, October). Optimal design of reconfigurable manufacturing systems. In *IEEE international conference on systems, man and cybernetics* (Vol. 3, pp. 6-pp). IEEE.
7. Pandarinath Potluri, Santhosh Kumar Rajamani, V. Brindha Devi, R. Sampath, Rajeev Ratna Vallabhuni, B. Mouleswararao, Bharathababu. K, Suraya Mubeen, "INTEGRATED SPECTRAL AND PROSODY CONVERSION WITH VOCODER OF VOICE SYNTHESIZER FOR HUMAN LIKE VOICE USING DEEP LEARNING TECHNIQUES," The Patent Office Journal No. 52/2022, India. Application No. 202241073323 A.
8. Li, Z., Janardhanan, M. N., Tang, Q., & Nielsen, P. (2018). Mathematical model and metaheuristics for simultaneous balancing and sequencing of a robotic mixed-model assembly line. *Engineering Optimization*, *50*(5), 877-893.
9. Borba, L., Ritt, M., & Miralles, C. (2018). Exact and heuristic methods for solving the robotic assembly line balancing problem. *European Journal of Operational Research*, *270*(1), 146-156.
10. Öztürk, C., Tunalı, S., Hnich, B., & Örnek, M. A. (2013). Balancing and scheduling of flexible mixed model assembly lines. *Constraints*, *18*, 434-469.
11. Rakesh Bharati, Sourabh jain, Prathiba Jonnala, Rishikesh Mishra, Meenu Singh, Rajeev Ratna Vallabhuni, Yadavalli. S. S. Sriramam, R. V. S. Lalitha, "MULTI-TASK MULTI-KERNEL LEARNING TECHNIQUE TO ASSESS AND CLASSIFY BIO AND PSYCHOLOGICAL SIGNALS," The Patent Office Journal No. 47/2022, India. Application No. 202211066338 A.
12. Liu, X., & Xu, Q. (2012). On signal selection for visibility enhancement in trace-based post-silicon validation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, *31*(8), 1263-1274.
13. Ma, J., Zuo, G., Loughlin, K., Cheng, X., Liu, Y., Eneyew, A. M., ... & Kasikci, B. (2020, March). A hypervisor for shared-memory FPGA platforms. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems* (pp. 827-844).
14. Merlini, M. A., Poy, I., & Chow, P. (2021, February). Interactive debugging at ip block interfaces in fpgas. In *The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays* (pp. 138-144).
15. Luo, T., Liu, S., Li, L., Wang, Y., Zhang, S., Chen, T., ... & Chen, Y. (2016). DaDianNao: A neural network supercomputer. *IEEE Transactions on Computers*, *66*(1), 73-88.
16. Luong, M. T., Pham, H., & Manning, C. D. (2015). Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*.
17. Lv, P., Liu, W., & Li, J. (2020, December). A FPGA-based accelerator implementaion for YOLOv2 object detection using Winograd algorithm. In *2020 5th International Conference on Mechanical, Control and Computer Engineering (ICMCCE)* (pp. 1894-1898). IEEE.
18. Farooq, U., Marrakchi, Z., Mehrez, H., Farooq, U., Marrakchi, Z., & Mehrez, H. (2012). FPGA architectures: An overview. *Tree-Based Heterogeneous FPGA Architectures: Application Specific Exploration and Optimization*, 7-48.
19. DR.T.NALLUSAMY, Dr. V.Kannan, Felipe De Castro Dantas Sales, Mr.J Logeshwaran, Mr. Rajeev Ratna Vallabhuni, Dr. SAYYED MATEEN, Ms.M.DHARANI, Mr. CH. Mohan Sai Kumar, Sanjesh Kumar, Mansi Singh, DR. SANDEEP KUMAR, PROF.DR.YEGNANARAYANAN VENKATARAMAN, "The detection of Varied EEG pattern Signal For Chronic Migraine Patients Using Machine Learning Approach," The Patent Office Journal No. 47/2022, India. Application No. 202241065256 A.
20. Aridhi, E., Popescu, D., & Mami, A. (2021). FPGA based co-design of a speed fuzzy logic controller applied to an

autonomous car. *International Journal of Reconfigurable and Embedded Systems (IJRES)*, *10*(3), 195-211.

21. Iyer, N., Anandmohan, P. V., Poornaiah, D. V., & Kulkarni, V. D. (2011, November). Efficient hardware architectures for AES on FPGA. In *International Conference on Computational Intelligence and Information Technology* (pp. 249-257). Berlin, Heidelberg: Springer Berlin Heidelberg.
22. John K-H, Tiegelkamp M. IEC 61131-61133: programming industrial automation systems.. 2001. https://doi.org/10.1007/978-3-662-07847-1.
23. Plaza, I., Medrano, C., & Blesa, A. (2006). Analysis and implementation of the IEC 61131-3 software model under POSIX real-time operating systems. *Microprocessors and Microsystems*, *30*(8), 497-508.
24. Thramboulidis, K. (2012, September). Towards an Object-Oriented extension for IEC 61131. In *Proceedings of 2012 IEEE 17th International Conference on Emerging Technologies & Factory Automation (ETFA 2012)* (pp. 1-8). IEEE.
25. Tang, U., Krezger, H., & LonnerbyRakob. (2024). Design and validation of 6G antenna for mobile communication. *National Journal of Antennas and Propagation, 6*(1), 6–12.
26. Bhowmik, S., Majumder, T., & Bhattacharjee, A. (2024). A Low Power Adiabatic Approach for Scaled VLSI Circuits. Journal of VLSI Circuits and Systems, 6(1), 1–6. https://doi.org/10.31838/jvcs/06.01.01
27. Antoniewicz, B., & Dreyfus, S. (2024). Techniques on controlling bandwidth and energy consumption for 5G and 6G wireless communication systems. *International Journal of Communication and Computer Technologies*, *12*(2), 11-20. https://doi.org/10.31838/IJCCTS/12.02.02
28. Sathish Kumar, T. M. (2023). Wearable sensors for flexible health monitoring and IoT. *National Journal of RF Engineering and Wireless Communication, 1*(1), 10-22. https://doi.org/10.31838/RFMW/01.01.02
29. Geetha, K. (2024). Advanced fault tolerance mechanisms in embedded systems for automotive safety. *Journal of Integrated VLSI, Embedded and Computing Technologies, 1*(1), 6-10. https://doi.org/10.31838/JIVCT/01.01.02
30. Borhan, M. N. (2025). Exploring smart technologies towards applications across industries. *Innovative Reviews in Engineering and Science, 2*(2), 9-16. https://doi.org/10.31838/INES/02.02.02
31. Velliangiri, A. (2024). Security challenges and solutions in IoT-based wireless sensor networks. *Journal of Wireless Sensor Networks and IoT, 1*(1), 8-14. https://doi.org/10.31838/WSNIOT/01.01.02
32. Uvarajan, K. P. (2024). Advanced modulation schemes for enhancing data throughput in 5G RF communication networks. *SCCTS Journal of Embedded Systems Design and Applications, 1*(1), 7-12. https://doi.org/10.31838/ESA/01.01.02
33. Uvarajan, K. P. (2024). Integration of artificial intelligence in electronics: Enhancing smart devices and systems. *Progress in Electronics and Communication Engineering, 1*(1), 7–12. https://doi.org/10.31838/PECE/01.01.02