IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)

An International Open Access, Peer-reviewed, Refereed Journal

Review Of Partial Reconfiguration Techniques For Fpga Acceleration

[1]Swayam Gupta, [2]Dr. Swapnil S N, [3]Dr. Pavithra G, Akshar Vastarpara

[1]Student, [2]Assistant Professor, [3]Associate Professor, [1]Dept of ECE, DSCE,
[1]Dept of ECE, DSCE, Bengaluru, India

Abstract: Field-Programmable Gate Arrays (FPGAs) offer significant performance and energy efficiency benefits, but their adoption is often hindered by long compilation times, which impede rapid development cycles and design space exploration. Traditional monolithic compilation flows optimize the entire design globally but require full recompilation even for minor changes. Partial Reconfiguration (PR) offers a promising avenue to address this challenge by enabling independent compilation and reconfiguration of specific design modules. This paper reviews three key research contributions that explore the use of PR, often coupled with overlay networks and high-level synthesis (HLS), to achieve faster parallel and incremental FPGA compilation. We synthesize the findings from Park et al. (2018), Xiao et al. (2019), and Xiao et al. (2022), highlighting the evolution of PR-based compilation strategies, the development of supporting tool flows (PRflow, HiPR), and the demonstrated reductions in compilation latency, particularly for incremental design changes.

Index Terms **- FPGA, Compilation Time, Partial Reconfiguration (PR), Incremental Compilation, Parallel Compilation, High-Level Synthesis (HLS), Overlay Networks.**

Introduction

Field-Programmable Gate Arrays (FPGAs) are increasingly utilized in diverse computing domains, including cloud data centers, embedded systems, and high-performance computing, owing to their hardware programmability and potential for significant application acceleration. However, a major barrier to the widespread adoption and efficient utilization of FPGAs is the protracted compilation process required to map a high-level design description to a physical bitstream. This High-Level Synthesis (HLS) to bitstream flow can take hours, starkly contrasting with the rapid compile-debug cycles common in software development. The monolithic nature of conventional FPGA compilation, where the entire design is synthesized, placed, and routed globally, is a primary contributor to these long turnaround times. Even small modifications often necessitate a complete recompilation, hindering iterative development, debugging, and thorough design-space exploration.

Partial Reconfiguration (PR) technology, supported by modern FPGAs, allows specific regions of the device to be reconfigured independently while the rest of the design remains operational. While traditionally used for runtime adaptability or reducing device area footprint, PR also presents a compelling opportunity to decompose the monolithic compilation problem. By dividing a design into static and reconfigurable partitions, PR can enable separate compilation strategies.

This paper reviews three significant works that investigate and develop methodologies leveraging PR to accelerate FPGA compilation:

1. Park et al. present an early case for using PR combined with a packet-switched network to enable faster parallel and incremental compilation.

2. Xiao et al. introduce PRflow, a tool flow automating separate compilation for HLS-defined building blocks using PR and an overlay network.

3. Xiao et al. propose HiPR, a framework that integrates PR concepts directly with HLS, allowing C/C++ level definition of reconfigurable functions and automating the flow for fast incremental compilation using modern PR features.

By synthesizing the approaches and findings of these papers, this review aims to provide a cohesive overview of how PR-based techniques can significantly reduce FPGA compilation bottlenecks.

I. BACKGROUND: FPGA COMPILATION AND PARTIAL RECONFIGURATION

A. Traditional FPGA Compilation:

The standard FPGA compilation flow involves several stages:

1. Synthesis: Translating a hardware description (e.g., Verilog, VHDL) into a netlist of logic gates and primitives.
2. Implementation:
   Optimization: Logic optimization on the netlist.
   Placement: Assigning logic elements to physical resources (e.g., LUTs, FFs, DSPs) on the FPGA fabric.
   Routing: Connecting the placed elements using the FPGA's programmable interconnect resources.
3. Bitstream Generation: Creating the configuration file (bitstream) that programs the FPGA.

The placement and routing steps are particularly time-consuming, often dominating the overall compilation time, especially for large and complex designs. These steps typically involve solving NP-hard optimization problems using heuristic algorithms. The monolithic nature means the entire design space is considered globally, which, while potentially yielding high-quality results (e.g., timing, density), makes the process slow and difficult to parallelize or incrementally update effectively.

B. Partial Reconfiguration (PR):

PR allows designated portions of an FPGA (Reconfigurable Regions, RRs, or p-blocks) to be reprogrammed with new partial bitstreams while the static logic outside these regions continues to operate uninterrupted. This requires defining physical boundaries for the RRs and managing the interfaces between the static and reconfigurable logic. Modern PR flows, like Xilinx's Dynamic Function eXchange (DFX), offer features like hierarchical regions and abstract shells to better isolate RRs and potentially speed up the implementation of individual modules.

C. Related Concepts:

Out-of-Context (OoC) Compilation: Vendor tools often support OoC flows, allowing individual IP blocks or modules to be synthesized independently before being integrated into a larger design for monolithic place and route. This aids team-based development but doesn't inherently accelerate the final, time-consuming implementation stages for the full chip.

High-Level Synthesis (HLS): HLS tools allow designers to describe hardware functionality using high-level languages like C, C++, or OpenCL, raising the abstraction level and improving productivity [4]. However, the output of HLS still feeds into the traditional, slow backend compilation flow.

II.PR-BASED APPROACHES FOR FAST COMPILATION

The reviewed papers leverage PR not just for runtime flexibility but specifically to tackle the compilation time bottleneck by enabling separate and incremental implementation.

A. Early Concepts and Motivation (Park et al. [1])

Park et al. [1] make the case for a "divide-and-conquer" strategy using PR to reduce FPGA compilation time, particularly the post-synthesis place and route stages. They propose partitioning the FPGA into multiple, independent PR regions (p-blocks) interconnected by a lightweight, packet-switched overlay network, specifically a Butterfly Fat Tree (BFT).

Methodology: The design is decomposed into leaf blocks (computational modules) mapped to individual p-blocks. The BFT network, potentially residing in its own PR region or the static region, handles communication between leaf blocks. This physical and logical isolation allows:

- Parallel Compilation: All leaf blocks can be implemented (placed and routed within their p-blocks) concurrently, for instance, on cloud or multi-core resources.
- Incremental Compilation: If only one leaf block changes, only that specific block needs to be re-implemented, avoiding a full-chip recompilation.
- Compute Model: A dataflow streaming model (Kahn Process Network style) is assumed, accommodating variable latencies introduced by the network.

Findings: Preliminary experiments on a Xilinx ZU9EG FPGA, using an array of MicroBlaze processors as leaf blocks, demonstrated that implementing a single leaf block (approx. 2700-3900 LUTs) took around 7 minutes, compared to over 30 minutes for a monolithic, full-chip build. Parallelizing the implementation of all leaf blocks reduced the effective full-chip implementation time to match the single-block time (7 minutes). This work established the potential of the PR-plus-network approach for significant compilation time reduction, while acknowledging trade-offs like area overhead for the network and potential fragmentation within p-blocks.

B. Toolflow Implementation and Evaluation (Xiao et al. [3])

Building upon the concept presented by Park et al. [1], Xiao et al. [3] developed and evaluated `PRflow`, a more concrete tool flow aimed at automating this separate compilation strategy, specifically targeting designs described using HLS.

Methodology: `PRflow` uses PR and an overlay packet-switched network (similar to [1]) to separate the HLS-to-bitstream compilation for individual components (operators or building blocks). It automates the process of mapping HLS-generated modules to PR regions. The flow supports both

Incremental Compilation: Recompiling only a single modified component.

Parallel Compilation: Compiling all components concurrently.

Implementation: The flow was implemented to work on top of commercial Xilinx Vivado tools and also explored using open-source tools (Yosys/VPR with Project X-Ray).

Findings: Mapping the Rosetta HLS benchmarks to a Xilinx XCZU9EG using `PRflow` on top of Vivado reduced compilation times significantly. For example, average times dropped from 42 minutes (monolithic) to 12 minutes (parallel separate compilation). One case saw a reduction from 160 minutes to 18 minutes. Preliminary results with open-source tools suggested potential for even faster times (under 5 minutes, some under 2 minutes). The work highlighted limitations in commercial tools not optimized for this usage pattern but demonstrated practical feasibility and substantial speedups.

C. High-Level Synthesis Integration and Automation (Xiao et al. [4])

Xiao et al. [4] further refined the approach with `HiPR` (High-level Partial Reconfiguration), focusing explicitly on bridging the gap between HLS design methodologies and low-level PR flows, aiming for greater automation and usability, particularly for software developers using FPGAs.

Methodology: `HiPR` allows developers to designate C/C++ functions as reconfigurable modules using pragmas directly in the HLS source code (`#pragma HLS PR`). This high-level specification drives the PR flow. Key features include:

Automated Floorplanning (`HiPlanner`): An integrated floorplanner (using Simulated Annealing or optionally MILP) automatically generates PR region definitions (sizes and locations) based on resource estimates from HLS/synthesis and device constraints. Users can provide resource scaling factors via pragmas to accommodate future growth.

Modern PR Features: Leverages Xilinx DFX features like abstract shells. Abstract shells create lightweight contexts for each PR region, significantly speeding up the place-and-route process during incremental compiles compared to loading a full static design checkpoint.

Tool Integration: Designed to work within the modern Xilinx Vitis development environment.

Compute Model: Assumes a Kahn Process Network model with stream interfaces, compatible with HLS dataflow paradigms.

Findings: Evaluated on Rosetta benchmarks using an Alveo U50 card, `HiPR` demonstrated significant incremental compilation speedups. When modifying a single PR function, recompilation times were reduced by **3x to 10x** compared to the standard Vitis incremental flow (which still performs full place-and-route). Median incremental compile times for individual operators were around 11 minutes. This acceleration was achieved primarily by confining the time-consuming place-and-route step to only the small, modified PR region using its abstract shell. The initial compilation (including overlay generation) incurred some overhead (15-67%) compared to the standard flow, but this one-time cost enables subsequent fast incremental turns. Performance (frequency, runtime) was generally maintained or slightly improved compared to standard Vitis flows.

III.SYNTHESIS AND DISCUSSION

The three papers demonstrate a clear progression in leveraging PR for FPGA compilation acceleration. Park et al. [1] established the foundational concept and feasibility. Xiao et al. [3] provided `PRflow`, a functional tool flow demonstrating significant gains with HLS components using both commercial and open-source backends. Xiao et al. [4] advanced the state-of-the-art with `HiPR`, tightly integrating the PR concept with HLS design practices through C-level pragmas, automated floorplanning, and the use of modern DFX features like abstract shells for dramatically faster *incremental* compiles within the Vitis ecosystem.

Key Themes and Evolution:

Target: Shift from RTL/generic blocks [1] towards HLS-generated components [3], [4], reflecting the increasing importance of HLS.

Automation: Increased automation from manual/scripted setup [1] to more automated flows in `PRflow` [3] and highly automated floorplanning and PR region definition based on HLS pragmas in `HiPR` [4].

Incremental Focus: While parallel compilation offers benefits for initial builds, the most significant speedups highlighted, especially in `HiPR` [4], are for incremental compilation, directly addressing the common edit-compile-debug loop bottleneck.

Technology Leverage:`HiPR` [4] explicitly utilizes newer vendor PR features (DFX, abstract shells) which are crucial for minimizing the overhead associated with implementing individual PR regions.

Interconnect: The use of an overlay network (BFT in [1]) is presented as a way to provide flexible connectivity between separately compiled blocks, decoupling them from a complex static routing structure. `HiPR` focuses more on direct stream connections managed within the Vitis/DFX framework.

Trade-offs:
Area Overhead: Overlay networks [1], [3] consume resources. PR itself can lead to resource fragmentation within p-blocks [1].

Initial Setup: Generating the initial PR design (the overlay or static portion) can incur a time overhead compared to a standard monolithic first compile, as seen in `HiPR` [4]. However, this is amortized over subsequent incremental builds.

Design Constraints: These approaches often assume a dataflow or streaming compute model to tolerate network latencies and interface cleanly with PR boundaries.

The consistent result across these studies is that decomposing the FPGA design using PR allows the time-consuming place-and-route stages to operate on much smaller, isolated problems, leading to substantial reductions in compilation time, especially for the incremental changes typical during development and debugging. The integration with HLS and automation provided by `HiPR` makes this approach more accessible to software-oriented FPGA developers.

IV. CONCLUSION

Long FPGA compilation times remain a significant challenge, hindering productivity and limiting the adoption of FPGA technology. The research reviewed here [1], [3], [4] compellingly demonstrates that Partial Reconfiguration, combined with techniques like overlay networks, separate compilation flows, HLS integration, and automated floorplanning, offers an effective strategy to mitigate this bottleneck. By enabling both parallel compilation of independent modules and, more critically, fast incremental recompilation of only modified components, these approaches can reduce compilation times from hours to minutes. Frameworks like `PRflow` and especially `HiPR` show the practical realization of these concepts, integrating them with commercial tools (Vivado, Vitis) and HLS design flows. While trade-offs exist, the significant acceleration of the edit-compile-debug cycle achieved through these PR-based methodologies represents a vital step towards making FPGA development more agile and accessible.

REFERENCES

[1] D. Park, Y. Xiao, N. Magnezi, and A. DeHon, "Case for Fast FPGA Compilation using Partial Reconfiguration," in *2018 International Conference on Field-Programmable Logic and Applications (FPL)*, Dublin, Ireland, Aug. 2018, pp. 235–238. doi: 10.1109/FPL.2018.00047.
[2] E. Chung et al., "Serving DNNs in real time at datacenter scale with project brainwave," *IEEE Micro*, vol. 38, no. 2, pp. 8–20, Mar./Apr. 2018. doi: 10.1109/MM.2018.112130020. (Note: Example reference showing FPGA usage context, cited in [4])
[3] Y. Xiao et al., "Reducing FPGA Compile Time with Separate Compilation for FPGA Building Blocks," in *2019 International Conference on Field-Programmable Technology (ICFPT)*, Tianjin, China, Dec. 2019, pp. 153–161. doi: 10.1109/ICFPT47387.2019.00026.
[4] Y. Xiao, A. Hota, D. Park, and A. DeHon, "HiPR: High-level Partial Reconfiguration for Fast Incremental FPGA Compilation," in *2022 32nd International Conference on Field-Programmable Logic and Applications (FPL)*, Belfast, United Kingdom, Aug. 2022, pp. 70–78. doi: 10.1109/FPL57034.2022.00022.
[5] Xilinx, Inc., *UG909: Vivado Design Suite User Guide: Dynamic Function eXchange*, UG909 (v2021.1), June 2021. [Online]. Available: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2021_1/ug909-vivado-partial-reconfiguration.pdf