2022 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES) | 978-1-6654-7296-8/22/$31.00 ©2022 IEEE | DOI: 10.1109/CASES55004.2022.00019

Work-in-Progress: An Open-Source Platform for Design and Programming of Partially Reconfigurable Heterogeneous SoCs

Biruk B. Seyoum
Computer Science
Columbia University
New York, USA
biruk@cs.columbia.edu

Davide Giri
Computer Science
Columbia University
New York, USA
davide_giri@cs.columbia.edu

Kuan-Lin Chiu
Computer Science
Columbia University
New York, USA
chiu@cs.columbia.edu

Luca P. Carloni
Computer Science
Columbia University
New York, USA
luca@cs.columbia.edu

***Abstract*—Dynamic partial reconfiguration (DPR) enables the design and implementation of flexible, scalable and robust adaptive systems. We present an FPGA-based DPR flow for partially reconfigurable heterogeneous SoCs that uses an incremental compilation technique to reduce the total FPGA compilation time.**

I. Introduction

By providing the capability to swap only a portion of the logic on the FPGA at runtime, dynamic partial reconfiguration (DPR) unlocks a great potential for implementing complex adaptive systems. Furthermore, it can be used to expand the space of system design for reconfigurable heterogeneous system-on-chip (SoC) architectures, which combine general-purpose processors with multiple domain-specific hardware accelerators. With the progress in semiconductor technology, SoCs hosts more and more heterogeneous components, making integration very challenging. Addressing these challenges requires the adoption of system-level design approaches [2]. In recent years, several open-source platforms have been proposed to support the system-level design of SoC architectures [1], [4], [5]. To date, however, most of these platforms target FPGA technologies only for rapid prototyping or functional verification purposes, without employing DPR-based design methodologies. This leaves behind the opportunity to use DPR for improving system performance and optimizing the FPGA compilation time (synthesis, place and route, and bitstream generation runtime) by using CAD-tool parallelism.

Realizing a platform that automates the integration of partially reconfigurable heterogeneous SoCs while improving FPGA compilation time demands a system-level design approach that addresses questions related to: *(i)* defining a system architecture that is compliant with DPR-related design constraints, *(ii)* creating algorithms that automate and parallelize the DPR FPGA flow using commercial CAD tools, and *(iii)* extending the heterogeneity of the architecture down to the FPGA implementation to enable a fine-grained incremental compilation of the design.

To address these challenges, we developed a DPR-based system-level design flow for partially reconfigurable heterogeneous SoCs. Our flow uses the heterogeneous tile-based distributed architecture of the open-source platform ESP [3], [5] as a baseline, and augments its architecture. Moreover, it introduces a novel DPR design flow that optimizes the bitstream generation runtime. This work-in-progress paper presents the *incremental flow* aspect of our approach. Our flow integrates an algorithm that enables a find-grained FPGA compilation of the design as well as a parallelization of the FPGA CAD flow, to reduce the total FPGA compilation runtime.

II. The Incremental Flow

ESP is an open-source research platform for heterogeneous SoC design and programming [3], [5]. It combines a scalable tile-based architecture and a flexibile methodology to integrate processors and loosely-coupled accelerators on a single chip. Our incremental flow for DPR focuses on supporting the introduction of reconfigurable accelerator tiles in ESP, while all the other tiles remain as *static parts* of the SoC design. Our flow introduces a fine-grained FPGA compilation that is aimed at reducing the recompilation runtime during iterative FPGA implementations, where only some of the reconfigurable tiles are modified in each iteration. The incremental flow detects the presence of newly modified reconfigurable tiles in the SoC design and generates new partial bitstreams only for them. The flow also introduces a novel approach for an on-demand parallelism during the synthesis and place and route (P&R) stages. While our flow exploits the out-of-context (OoC) synthesis mode that is offered by Vivado, the P&R of reconfigurable tiles is performed in parallel, on top of a pre-implemented static part, by using separate Vivado instances.

The incremental flow utilizes a custom algorithm, which is described with a high-level pseudo-code in Listing 1. In each design iteration, the algorithm starts with a comparison of the SoC configurations between the current and previous design runs to determine if the static part or any of the reconfigurable tiles have been modified. The changes in the static and reconfigurable parts are mainly detected by parsing the name, type, and additional parameters of the tiles inside both configurations. If a change is detected in the static part (e.g., tiles in the static region are modified or the total number of tiles in the SoC is changed), then this invalidates all the previous design runs and the algorithm invokes the full flow to implement the entire SoC (line 16). Instead, if the changes are only limited to reconfigurable tiles, then the flow performs

2643-1726/22/$31.00 ©2022 IEEE
DOI 10.1109/CASES55004.2022.00019

Listing 1. pseudo-code for our incremental compile flow

```
1  //struct variables that hold the esp SoC configuration
2  struct new_config, old_config;
3  int modified_tiles;
4  //implementation mode {PARALLEL or Serial}
5  int mode = PARALLEL;
6
7  //check how many tiles are modified in the current iteration
8  modified_tiles = compare_configs(new_config, old_config);
9
10 //if no tile is modified, exit
11 if(modified_tiles == 0)
12    return 0;
13
14 //check if the SoC configuration is modified
15 if(is_static_modified(new_config, old_config) {
16    goto: run_full_flow(mode);
17 }
18 else {
19    //synthesize newly modified tiles
20    synth_modified_acc_tiles(new_config, mode);
21 }
22
23 //extract resource consumption of tiles from vivado report
24 parse_resource_consumption(new_config, old_config);
25
26 //check if the modified accelerators require more resource
27 //than their predecessors; if so, floorplanning again
28 for(int i = 0; i < modified_tiles; i++) {
29    if(!chk_resource_util(new_config, old_config, i)){
30       floorplan();
31       goto: implement_all_acc_tiles(mode);
32    }
33 }
34 goto: implement_only_modified_acc_tiles(mode);
```

a parallel OoC synthesis of all the modified reconfigurable accelerators (line 20).

After the synthesis is complete, the resource consumption of the new accelerators is extracted (line 24) and is compared to the resources that are already allocated to the reconfigurable tile where the new accelerators are to be placed (line 29). If enough resources are available in the host tiles, then partial bitstreams are generated only for the modified accelerators by using an already placed and routed design checkpoint, which contains the unmodified part of the SoC (line 34). In the worst case, i.e., one or more of the newly modified accelerators consume more resources than what is available in the reconfigurable tiles, a new floorplanning design step is executed and partial bitstreams are generated both for the modified and unmodified accelerators (line 29 - line 31). Note that, even in this worst case, the algorithm in the incremental flow avoids re-synthesizing the static part of the design, which usually is the second most time consuming step after P&R. Our flow uses an open-source DPR floorplanning tool [6]. Besides its flexibility, the incremental flow offers an opportunity to save valuable synthesis and implementation time that would have otherwise been wasted in re-implementing the full SoC, even for minor changes during an iterative design.

III. Experimental Evaluation

To evaluate our incremental flow, we generated three SoC design instances, with 3x4, 3x5 and 4x5 tile configurations, respectively. In the SoCs, all the reconfigurable tiles were instantiated with MAC accelerators from the ESP accelerator suite. At each design iteration, an incremental percentage of the reconfigurable tiles was modified and the design runtime to regenerate the new partial bitstreams was recorded. To guarantee enough resources for the new accelerators in each tile and to reduce the design sensitivity to resource variations, the floorplan `pblocks` were generated with a 10% resources margin. The experiment was performed in two modes. In the first mode, the parallel compilation was disabled (denoted as *serial* in Fig. 1) and all modified tiles were recompiled using a single Vivado instance. This demonstrates the net speedup from our incremental compilation flow without any boost from parallel P&R. In the second mode, instead, the incremental flow was executed by also exploiting the CAD-level parallelism.

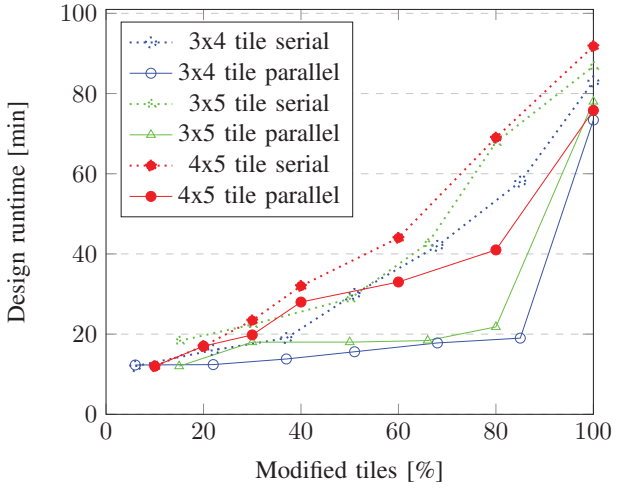

Fig. 1. Total design runtime to generate partial bitstreams for designs with incremental changes per iteration.

Fig. 1 reports the results of this experiment. Note that, without incremental compilation, the design must always be recompiled fully even for minor modifications. As expected, the design runs that were executed without parallelism take longer to complete as more and more tiles are modified. But even without parallelism, the incremental compilation keeps the design runtime to a minimum. In addition to this, enabling parallel implementation within the incremental flow significantly reduces the design runtime.

Acknowledgments. This work was supported in part by DARPA (C#: FA8650-18-2-7862) and in part by the NSF (A#: 1764000). The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of Air Force Research Laboratory and DARPA or the U.S. Government.

References

[1] A. Amid et al. Chipyard: Integrated design, simulation, and implementation framework for custom SoCs. *IEEE Micro*, 40(4):10–21, 2020.
[2] L. P. Carloni. From latency-insensitive design to communication-based system-level design. *Proc. of the IEEE*, 103(11):2133–2151, 2015.
[3] L. P. Carloni. The case for embedded scalable platforms. In *Proc. of the Design Automation Conf. (DAC)*, pages 1–6, 2016.
[4] C. Heinz et al. The TaPaSCo open-source toolflow. *Journal of Signal Processing Systems*, 93(5):545–563, 2021.
[5] P. Mantovani et al. Agile SoC development with Open ESP. In *Proc. of the Intl. Conf. on Computer-Aided Design (ICCAD)*, 2020.
[6] B. B. Seyoum et al. FLORA: floorplan optimizer for reconfigurable areas in FPGAs. *ACM Trans. Embed. Comput. Syst.*, 18(5s), Oct. 2019.