Circumventing Restrictions in commercial High-Level Synthesis Tools

Benjamin Carrion Schafer and Chaitali G. Sathe
The University of Texas at Dallas
Department of Electrical and Computer Engineering
{schaferb, chaitaligajanan.sathe}@utdallas.edu

***Abstract*—Many Software (SW) vendors limit the functionality of their product based on the version purchased. This trend has also carried over to Electronic Design Automation (EDA). For example, Field-Programmable Gate Array (FPGA) vendors make their Lite versions freely available to anyone, but charge for their full versions, e.g., Intel Quartus Prime Lite vs. Quartus Prime.**

Some High-Level Synthesis (HLS) tool vendors have started to do the same in order to appeal more to FPGA users who are more price conscious as opposed to the ASIC users. FPGA tools are typically free or very inexpensive and hence, it makes sense to have dedicated FPGA versions of their HLS tools. To enable this strategy some HLS vendors have put in place different control mechanisms to avoid anyone using their inexpensive FPGA version to target ASICs, as this would defeat their price discrimination strategy.

In this work, we review different strategies used by the HLS vendors and propose to the best of our knowledge the first technique to circumvent these. In particular we show how we can generate ASIC circuits with similar area and performance using the HLS Lite HLS version that only allows to target small FPGAs as compared to using the full ASIC HLS version.

I. Introduction

High-Level Synthesis (HLS) converts untimed behavioral descriptions (e.g., ANSI C, C++ or SystemC) into efficient hardware circuits specified in either Verilog or VHDL. It has been shown that raising the VLSI design abstraction from the RT-level to the behavioral level has many advantages like reducing the turn around time, allowing faster simulations and the ability to generate different types of hardware circuits from the same behavioral descriptions by simply setting different synthesis options [1].

One of the main problems with commercial HLS tools is the cost disparity between the ASIC and the FPGA vendors. FPGA vendors make their tools virtually free as they can price the cost into the FPGA unit cost, while traditional EDA tools vendors cannot. Thus, HLS tools from traditionally EDA vendors mainly target ASICs and are orders of magnitude more expensive, but because many ASIC designers also have to emulate and prototype their designs, these tools also target FPGAs. Thus, it makes sense for these HLS vendors to also offer dedicated FPGA versions to compete with the inexpensive FPGA vendors' tools. One significant advantage that these traditional vendors try to leverage is that their HLS tools can target FPGAs from different vendors while FPGA vendors only support their own FPGAs. The main problem that these vendors face is that the inexpensive HLS tool versions should limit the features offered and also only allow to target FPGAs and not ASICs.

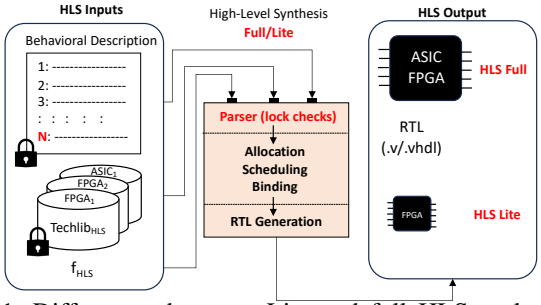

Fig. 1: Differences between Lite and full HLS tool version: (1) Code size, and (2) devices targeted.

Fig. 1 shows the differences between these two HLS tool versions (HLS Lite and HLS full)[1]. Both take as input the behavioral description to be synthesize, a target synthesis frequency (f_{HLS}) and a technology library (techlib) that contains the area and delay information for basic operators of different bitwidths like adders and multipliers. Removing some capabilities of the HLS tool is rather straight forward as the HLS tools are typically built in a modular fashion with separate binaries for each individual feature. The main problem is how to *limit* the functionality. These include allowing to target only certain FPGA families and no ASICs and also limiting the size of the circuit to be synthesized by e.g., limiting the numbers of size of the code that can be synthesized. This allows HLS vendors to fully discriminate the price of the product line up by providing a dedicated FPGA version as well as an additional basic versions that can only target smaller FPGAs.

Considering this, the main goal of this work is to circumvent these restrictions and provide an automated methodology to allow to synthesize ASICs and any FPGA using the most restrictive HLS tool version that only targets smaller FPGAs and allows to synthesize small circuits. We aim at exposing these weakness in order to make EDA vendors aware of them. In summary, the main contributions of this work are:

- Introduce a design flow that allows the use of restricted (lite) HLS tools that can only target small FPGAs to synthesize efficiently ASICs.
- Extend the flow to circumvent the circuit size limitation allowing to generate hardware designs of any size.
- Present a comprehensive evaluation show the effectiveness of our proposed flow compared to directly synthesizing the entire description using an unrestricted version of the tool.

[1]We will call these different HLS tool version HLS Lite and HLS Full.

979-8-3503-4859-0/DATE24/© 2024 EDAA

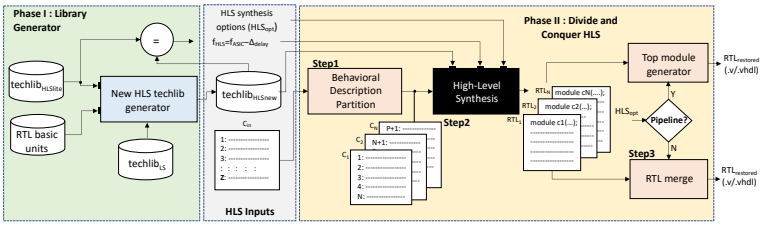

Fig. 2: Overview of complete proposed flow composed of two main phases: Phase I, generation of new technology library or frequency adjustment. Phase II, divide and conquer HLS flow.

II. Proposed Flow

In this work we assume as mentioned previously that the HLS tool locks the type of hardware devices that can be targeted and the size of the behavioral description that can be synthesized. Fig. 2 shows an overview of our complete proposed flow that is composed of two phases. **Phase I** generates the new HLS technology library of the FPGA family or ASIC not supported by the lite version of the HLS tool. **Phase II** then automatically partitions the behavioral description in portions that can be synthesized by the lite HLS tool and then merges the results. These individual descriptions are in turn synthesized (HLS) and the results then merged. We consider two cases here. In the first one the circuit has to be pipelined and second, it is not. In the pipelined case merging the different resultant circuits is easier as the different hardware designs can be directly connected when synthesized with the same initiation interval. In the non-pipelined case, the FSM and data paths generated by the HLS tools for the different partitions need to be merged into a single design.

The results of the flow is a newly restored RTL description ($RTL_{restored}$) that has to be as close as possible in terms of area and latency to the RTL case generated when the entire behavioral description is synthesized with the unlimited HLS tool version (RTL_{full}).

III. Experimental Results

We evaluate the effectiveness of our proposed flow by synthesizing different behavioral description from the S2CBench [2] benchmarks suite with the full version of the same tool (baseline). In this case CyberWorkBench v.6.1.1 from NEC is used [3] which commercializes different HLS tool versions as described in this work. In both cases we target the synthesis of these behavioral descriptions to a 45nm Nangate Open cell ASIC technology. Basically, we try to understand if we can efficiently target ASICs with the lite version of the HLS tool that only allows to synthesize small behavioral descriptions targeting smaller FPGA families. Our approach is shown to be effective if the generated RTL designs are similar in terms of area and performance (latency) to the direct ASIC synthesis of the behavioral description with the full HLS tool version.

Fig 3 summarizes the are overhead results. In all of the cases, we synthesize the benchmarks by maximizing the parallelism by fully unrolling looks and inlining functions (fully parallelized) and also re-synthesize the benchmarks fully

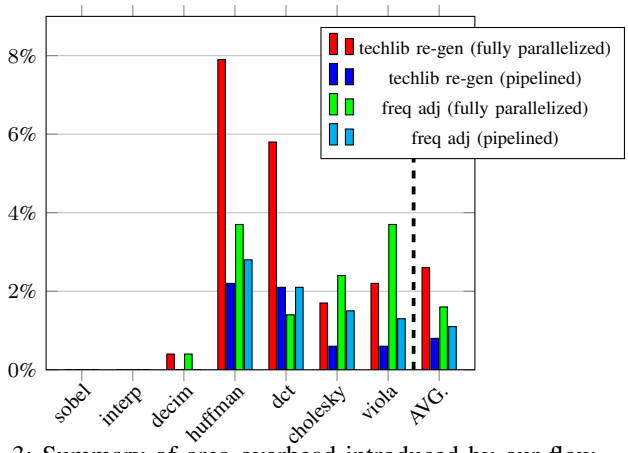

Fig. 3: Summary of area overhead introduced by our flow.

pipelining them setting the smallest Data Initiation Interval as possible (DII), which in all cases was 1 clock cycle. This allows us to fully characterized our proposed flow as we generate different RTLs depending on if the design is pipelined or not.

We can make the following observations from these experimental results:

Observation 1: In all cases, our proposed approach was able to generate a functional equivalent circuit to the baseline approach.

Observation 2: Our proposed approach did lead to area overheads for the larger benchmarks (last 4 benchmarks). On average the area increased by 2.6% and 1.6% case in the fully parallelized version and 0.8% and 1.1% in the pipelined case. We observed multiple reasons for these area overheads: (1) The HLS tool did automatic bithwidth optimizations of the internal signals. Partitioning the description limited the amount optimizations done. (2) Our approach shared FUs, but not registers. (3) Some partitions required to include redundant operations computed in previous partitions and hence, needed to be duplicated. The overheads were nevertheless not significant.

IV. Conclusion

In this work, we have introduced a method to circumvent the limitations used by some commercial HLS vendors to limit the size of the behavioral description that can be synthesized and the target hardware platform. This allows them to release different tool versions that appeal to different customer bases, e.g., FPGA users vs. ASIC users. We have shown that your proposed flow is able to efficiently synthesize large behavioral description targeting ASICs using a commercial HLS tool that only allows to target small FPGAs.

References

[1] B. Carrion Schaefer, *High-Level Synthesis Made Easy*. highX Technlogies, 2023. [Online]. Available: www.hlsbook.com

[2] B. Carrion Schafer and A. Mahapatra, "S2CBench:Synthesizable SystemC Benchmark Suite," *IEEE Embedded Systems Letters*, vol. 6, no. 3, pp. 53–56, 2014.

[3] NEC CyberWorkBench. (2023). [Online]. Available: www.cyberworkbench.com