LiveSynth: Towards an Interactive Synthesis Flow

Rafael Trapani Possignolo, Jose Renau
Dept. of Computer Engineering
University of California, Santa Cruz
http://masc.soe.ucsc.edu

Background:

➔ Synthesis is tedious and time consuming, especially during the timing/power closure cycle.
➔ This contrasts with rapid development techniques popular in software engineering.
➔ We expect designers productivity to improve with an interactive synthesis environment.

Model:

➔ **LiveSynth** targets interactive synthesis with feedback within a few seconds.
➔ LiveSynth allows designers to trigger synthesis more frequently and incrementally.
➔ LiveSynth flow is divided into two phases:
  ➔ **Interactive step:** gives feedback in under a few seconds, with high accuracy
  ➔ **Background step:** high effort optimization, when the designer is not making changes

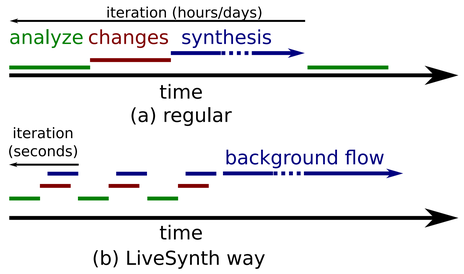

Fig 1. LiveSynth shifts the digital design paradigm to incremental changes, allowing for more interations per day.

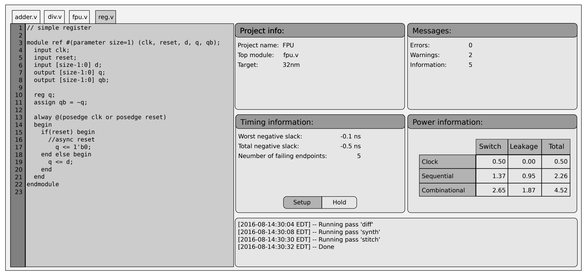

Fig 2. Mockup concept for LiveSynth. As the designer updates the code, new results are displayed.

Incremental Flow:

➔ LiveSynth automatically defines regions of a few thousand gates that are used as incremental grains.
➔ Invariant cones [1] are regions whose functionality do not change during synthesis and are used by LiveSynth.
➔ During the incremental step, only cones that were changed are re-synthesized.
➔ To avoid impact on QoR, if the critical path is hit, the neighbor regions are also synthesized.

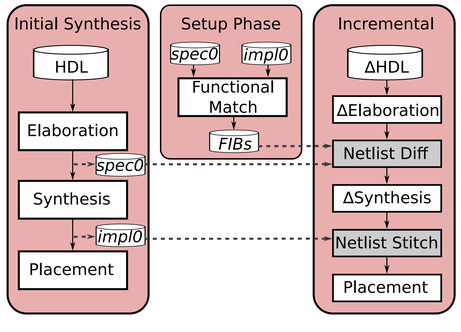

Fig 3. The initial synthesis is performed as usual, and the incremental step is performed when the designer changes the RTL.

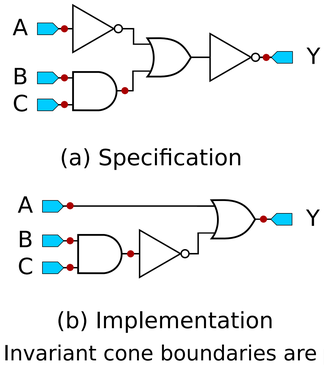

Fig 4. Invariant cone boundaries are present over digital designs and provide good granularity for incremental synthesis.

Setup:

➔ We implemented the incremental step of LiveSynth in Ruby.
➔ We used an in-house FPU verilog code as benchmark.
➔ 32 changes were added in randomly choosen locations, activated through `define` statements.
➔ LiveSynth was run on-top of a commercial flow and YOSYS [2], an open-source synthesis tool.

Results:

➔ The incremental step of LiveSynth achieves ~95% faster synthesis than a full run (Figures 5 and 6).
➔ There was no significant difference in Fmax between LiveSynth and full synthesis (Figures 7 and 8).

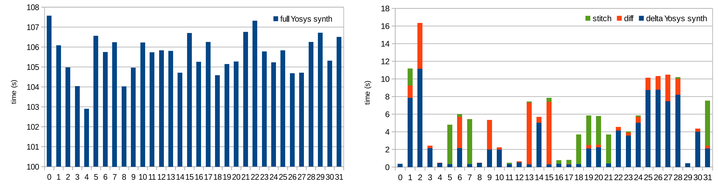

Fig 5. LiveSynth is built on top of third-party tools and is able to reduce runtime by ~94% compared to YOSYS.

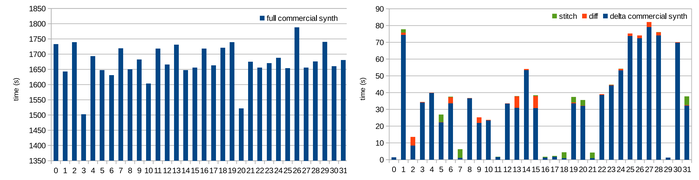

Fig 6. LiveSynth reduces runtime by ~96% compared to a commercial flow.

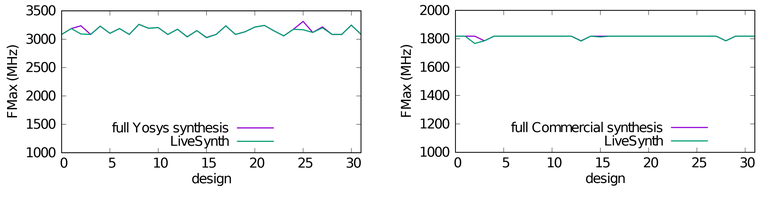

Fig 7. LiveSynth is able to deliver the same QoR as the full synthesis, with minor fluctuations.

Conclusion:

➔ The incremental step of LiveSynth reduces synthesis time by about 95% for incremental changes.
➔ LiveSynth shifts the paradigm to small, incremental changes and more iterations per day.
➔ We advocate for an interactive synthesis flow as a way to boost design productivity.

Future Work:

➔ Incremental back-end to further improve on feedback accuracy.
➔ Improve synthesis to reduce QoR impact.
➔ Further reduce synthesis area to reduce synthesis time in the outliers.
➔ FPGA target with further improvement on backend.

Acknowledgments:

This work was supported in part by the National Science Foundation under grants CNS-1059442-003, CNS-1318943-001, CCF-1337278, and CCF-1514284. Any opinions, findings, and conclusions or recommendations expressed herein are those of the authors and do not necessarily reflect the views of the NSF.

References:
[1] D. Chen and D. Singh, “Line-level incremental resynthesis techniques for fpgas,” in FPGA’11.
[2] C. Wolf. Yosys open synthesis suite. http://www.clifford.at/yosys/