Support for Partial Reconfiguration Regions Through Partition Pins and VPR Constrained Placement/Routing

Synopsis

SymbiFlow does not currently support partial reconfiguration regions. Partial reconfiguration regions are crucial to ongoing FPGA research including reducing verilog to bitstream compilation times through separate compilation. Two major hurdles to the support of partial reconfiguration regions are SymbiFlow and VPR support for restricted placement/routing and the ability to generate and upload a partial bitstream. I will be taking on the first of these two hurdles.

Benefits to Community

This work has the ability to have a huge impact on the entire FPGA community. One large factor holding back people from using FPGAs rather than software is the long edit, compile, debug cycle time, especially with large designs. Current vendor tools such as Xilinx Vivado do not have much support for speeding up compilation times because they are focussed on producing optimal designs. The ability to have compilation options that trade-off implementation quality and time will greatly decrease the edit, compile, debug cycle and make it easier for engineers to make use of FPGAs and hardware acceleration.

One of the most promising ways to decrease the compilation time is through separate compilation, essentially making use of a divide and conquer methodology to split up the design into a number of smaller pieces that are compiled separately. Not only does this provide speedup from being able to utilize multi core machines more effectively, but compiling smaller, restricted designs reduces the amount of routing checks and changes super-linearly.

In order to support this kind of separate compilation, SymbiFlow must support partial reconfiguration regions. Doing so will give researchers greater freedom to experiment with separate compilation than can be provided by current vendor tools.

Deliverables

May 4: Accepted GSoC 2020 students/projects are announced

May 4 - 31: Community Bonding Period

June 1 - June 7: Investigate SymbiFlow architecture file generation and VPR for restricting placement/routing (Required)

June 8 - June 28: Write code to support the restriction of placement/routing (Required)

June 29 - July 5: Document restricting placement/routing (Required) (Milestone 1)

July 6 - July 12: Investigate SymbiFlow architecture file generation to add manual partition/IO pins (Required)

July 13 - July 26: Write code to support manual partition/IO pins (Required)

July 27 - August 2: Document adding manual partition/IO pins (Required) (Milestone 2)

August 3 - August 9: Investigate SymbiFlow architecture file generation to automatically choose partition/IO pins (Optional)

August 10 - August 16: Write code to support automatic partition/IO pins (Optional)

August 17 - August 20: Document adding automatic partition/IO pins (Optional) (Milestone 3)

August 20 - August 24: The whole system should be working, ensure really does

August 24 -31: Wrap up their projects and submit final evaluation

September 8: Students passing GSoC 2020 are announced

Potential other work if time permits:

1. Investigate combining multiple restricted routings/partial reconfiguration regions into one bitstream
2. Write code to support combining multiple restricted routings/partial reconfiguration regions
3. Document combining multiple restricted routings/partial reconfiguration regions

Related Work

This concept is similar to the ROI initially used in project x-ray, but would support choosing any reasonable region on the chip and hopefully automatically generating partition pins (equivalent to synth IO pads).

There is also similar work to VtR issue #932 which discusses supporting placement constraints in VtR. In my mind there are trade offs between this method and pre-generating an ROI like miniature architecture file. Constraints in VtR is probably a more elegant solution, but at least without additional work would still load the entire architecture file and rr graph into memory, which is a significant portion of the compile time for large devices. In contrast, using something like an ROI would require a longer compile initially to generate the architecture file and rr graph, but VtR would only have to load in the smaller architecture file and this method would inherently constrain placement and routing because VtR does not have access to the rest of the FPGA.

This project also has some ability to work with fitting into the FPGA Tooling Common Interchange Format especially when defining how to place IO ports on the partial reconfiguration region.

Biographical Information

I am currently a Junior at the University of Pennsylvania studying computer engineering with an interest in computer architecture and FPGAs. I have previously used SymbiFlow while working in the Implementation of Computation group at the University of Pennsylvania under Professor Andre DeHon. I worked over summer 2019 on speeding up compilation for a monolithic design in SymbiFlow. This included implementing a binary rr_graph writer and reader as well as running a number of experiments to see how different SymbiFlow options traded-off compilation time and implementation quality. My work on a binary rr_graph writer and reader was accepted into the SymbiFlow VPR repository, and has since been improved by others who used a standardized binary format. My work with the Implementation of Computation group was included in a paper published at FPT19 titled “Reducing FPGA Compile Time with Separate Compilation for FPGA Building Blocks”.

Name and Contact Information

Andrew Butt
andrewb1999@gmail.com or butta@seas.upenn.edu
GitHub: andrewb1999
Mobile Phone: +1 (813) 817-1304
Home Address: 12125 Clear Harbor Drive, Tampa, FL 33626