BTP Presentation

Chinmay Purandare
14/12/2020

Agenda

1. RocketChip Generator
2. Rocket Core
3. BOOM
4. FireSim
5. Golden Gate
6. Further Work

RocketChip Generator

- It is written in Chisel and constructs a RISC-V-based platform
- It consists of a collection of parameterized chip-building libraries that we can use to generate different SoC variants
- It has created a plug and play like environment by standardizing the interfaces between connected various chip building libraries

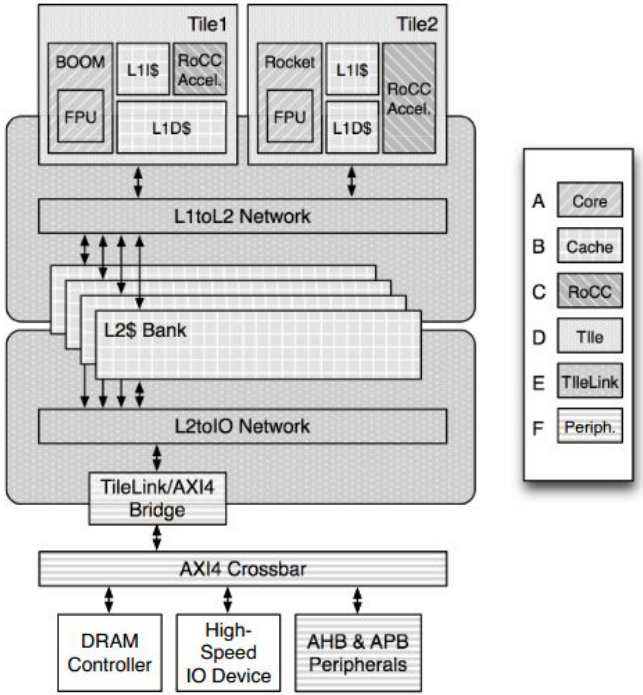

Figure 1: The Rocket Chip generator consists of the following sub-components: A) Core generator B) Cache generator C) RoCC-compatible coprocessor generator D) Tile generator E) TileLink generator F) Peripherals

Rocket Core

- RocketCore is a 5-stage in-order scalar core generator that implements RV32G and RV64G ISAs.
- It has an MMU which supports page-based virtual memory, a non-blocking data cache, and a front-end with branch prediction.
- It can also be thought of as a library of processor components.
- Several modules originally designed for Rocket are re-used by other designs, including the functional units, caches, TLBs and the page table walker.
- The Rocket Custom Coprocessor Interface (RoCC) facilitates decoupled communication between a Rocket processor and attached coprocessors.

BOOM

- BOOM is an out-of-order, superscalar RV64G core generator.
- It is written in 10k lines of Chisel code, which combines core modules written for BOOM with instantiating multiple modules from the RocketChip repository.
- It supports full branch speculation.
- It uses an aggressive load/store unit, which allows loads to execute out-of-order with respect to stores and other loads, which can also forward store data in queues to dependant loads.

FireSim

- Uses decoupled FAME transforms provided by the MIDAS/Strober frameworks to translate the server designs written in Chisel into RTL with decoupled I/O interfaces to use during simulation
- Each target cycle, the transformed RTL on the FPGA expects a token on each input interface to supply input data for that target cycle and produces a token on each output interface to feed to the rest of the simulated environment.
- A rate limiter is added which limits the bandwidth at runtime using a token-bucket algorithm.
- Since FireSim was simulating a single target design in each FPGA its resource utilization was very low
- This led to the development an additional “supernode” configuration, which simulates multiple complete target designs on each FPGA to provide improved utilization and scalability.

Golden Gate

- Golden Gate-generated simulators decouple the target-design clocks from all FPGA-host clocks. So one cycle in the target machine is simulated over a dynamically variable number FPGA clock cycles.
- We can run Structures in ASIC RTL that map poorly to FPGA logic with models that preserve the target RTL's behavior.
- This takes more host(FPGA) cycles but is resource efficient on the FPGA
- Since host and target clocks are decoupled we can host the components of the targets over multiple FPGAs
- So the targets simulated performance isn't based on the memory available in the host

Golden Gate

- Although the decoupling introduces overheads not present in an FPGA-prototype depending on the features implemented and optimizations applied.
- This might increase host resource use, decrease fmax, and decrease overall simulation throughput

What Golden Gate Actually Does

When a design is passed to Golden Gate:

1) It finds the target module and disconnects the module and connects the interface to the top level of the design
2) It then wraps the target interfaces to match the host interface definition
3) Once the target optimizations are done it generates the bridge modules and connects them to tokenized interfaces

Further Work

- We plan to examine from source how Golden Gate goes about compiling the hardware design, and see what additions can be made to it so as to make it perform more optimizations specific to the design
- Also, currently, Golden Gate only supports optimizing multi-ported memories in the target RTL
- We plan to add support to optimize other FPGA hostile structures in the RTL

In conclusion, after examining and understanding the working of the use cases mentioned, we are now working on studying and identifying FPGA hostile structures which can be optimized, as well as seeing what more optimizing can be made to Golden Gate.

Thank You