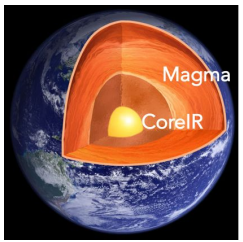
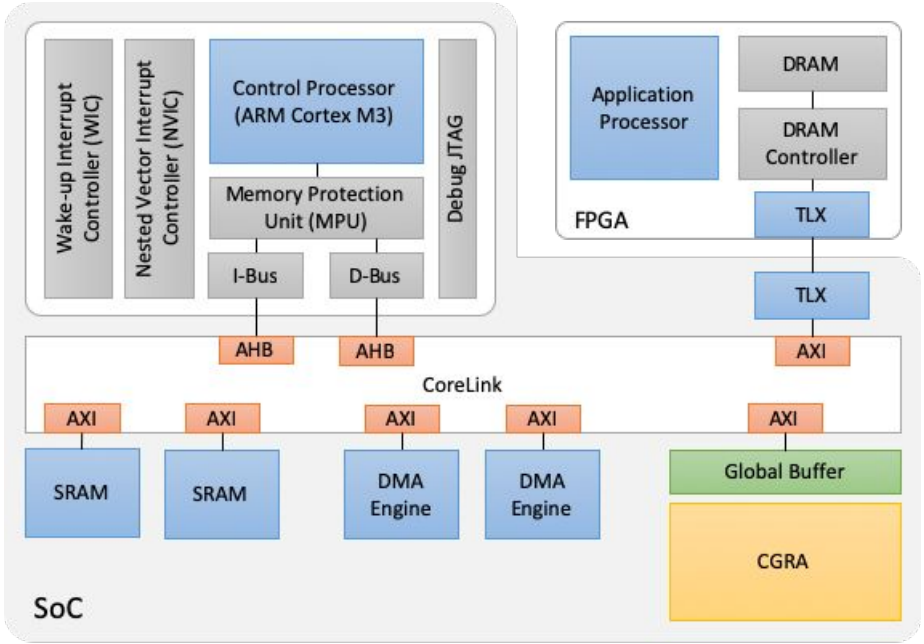
Garnet SoC

- We (almost) taped out our chip!
  - Have a physical design that is DRC and LVS clean, meets (relaxed) timing, and is functionally correct!
  - **Much** better unit tested than Jade
  - Apps run end-to-end on the SoC, with software exercising M3, DMA, TLX, AXI, global buffer, and CGRA
- Uses our DSLs to generate the hardware and the software toolchain!





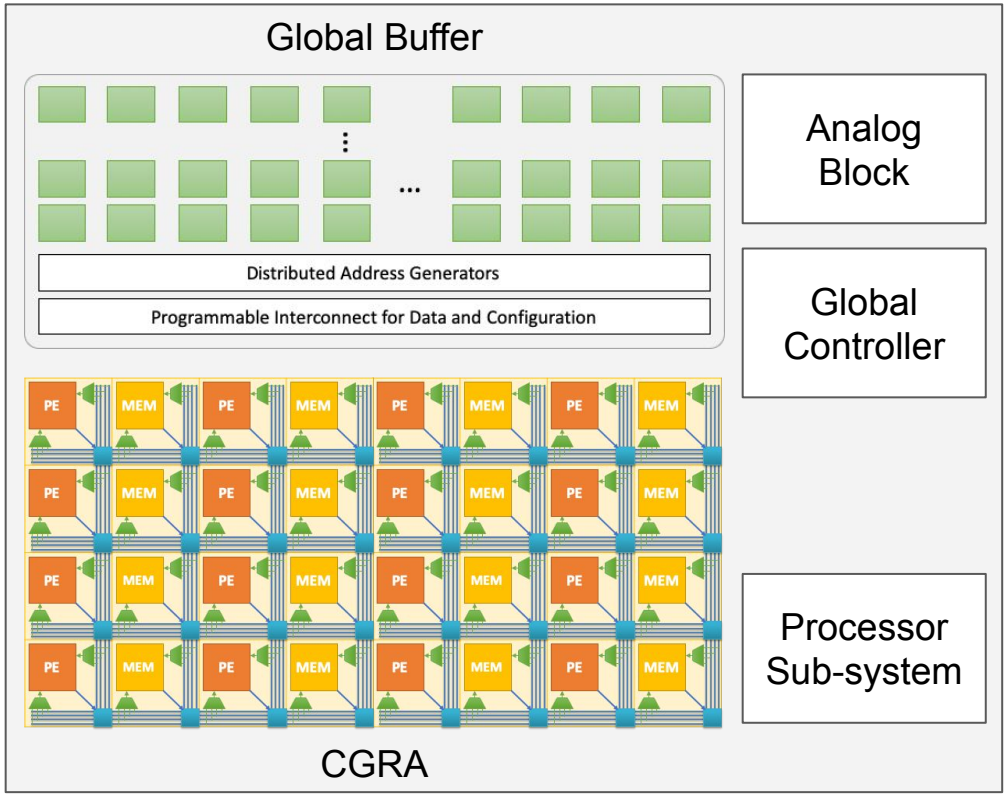
Agenda for today

- What did we learn from this tapeout
  - What went well?
  - Where can we improve?

- We will present four key areas where there is the biggest opportunity
  - **Physical Design using Gemstone and Virtual Tapeout** (Alex, Priyanka) 11:00 - 11:20
  - **Automatic Mapper** (Ross) 11:20 - 11:40
  - **Lake** (Joey) 11:40 - 12:00
  - **Halide Compiler for SoC** (Jeff) 12:00 - 12:20
  - Lunch 12:20 - 13:10
  - Feedback 13:10 - 13:30

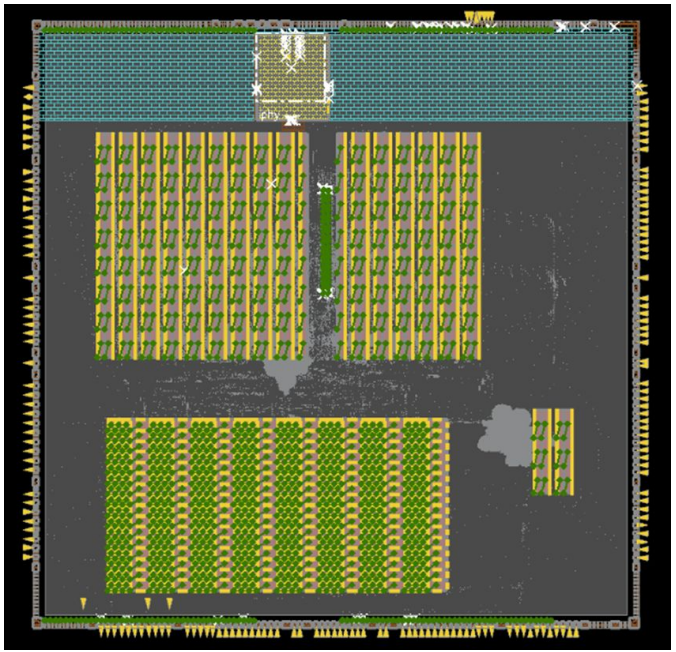
Garnet SoC

- 32x16 array of CGRA tiles
- 4 MB of SRAMs (256 individual macros)
- Analog block
- Processor sub-system
  - Cortex M3
  - A few SRAMs
  - Not large (1-2% of total chip area)



Overview of the Current Physical Flow

- Full P&R flow for each tile type (PE and Memory)
- Then essentially do everything else at the top-level
  - Manually place all SRAM macro arrays using flexible/parameterized procedures
  - Manually abut the tiles
  - No interconnect routing required because adjacent tiles are connected by abutment
  - Create soft regions for Global buffer, CGRA, and processor subsystem logic
  - Tool places and routes all cells for global buffer control, processor subsystem



What worked well

- We got a chip out
- Much denser CGRA layout with river routed global signals and fully abutted layout
  - But introduced complications for top level timing
- Physical design scripts much more reusable than before
  - Everything technology-dependent is a parameter (and all of these parameters are in one place in the scripts)
  - Several design-dependent parameters are calculated by introspecting the design and taking into account a small set of constraints: tile sizes, power stripe pitches, tile grid dimensions

Problems with Current Physical Flow

- Difficult to debug issues with a flattened top design
  - Flow needs to be more block oriented and staged for easier integration, timing closure, shorter design cycles
- Fundamental timing issues
  - Synthesis didn’t tell us much and interfaces made it hard to fix
- Not fully amenable to incremental logical design changes

- Not running the full flow early enough
  - Tools are complex
  - Innovus works hard to complete the steps even if the constraints are not fully specified/reasonable
  - Need to look at early indicators of downstream issues at each stage
- Reproducibility of the flow
  - Better version control of scripts, reproducibility of full environments

Problems with Current Physical Flow

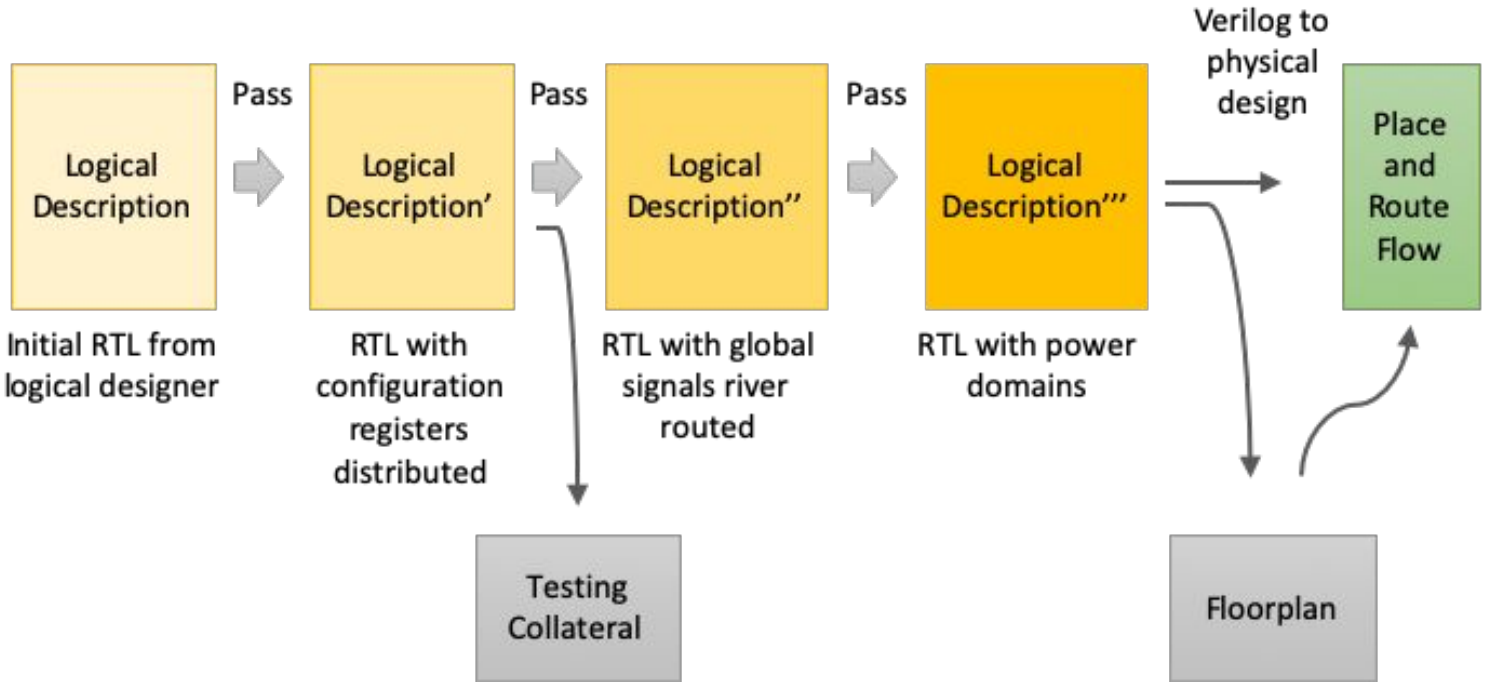
- Difficult to debug issues with a flattened top design
  - Flow needs to be more block oriented and staged for easier integration, timing closure, shorter design cycles
- Fundamental timing issues
  - Synthesis didn't tell us much and interfaces made it hard to fix
- Not fully amenable to incremental logical design changes

Staged generation of hierarchical physical flow with gemstone

- Not running the full flow early enough
  - Tools are complex
  - Innovus works hard to complete the steps even if the constraints are not fully specified/reasonable
  - Need to look at early indicators of downstream issues at each stage
- Reproducibility of the flow
  - Better version control of scripts, reproducibility of full environments

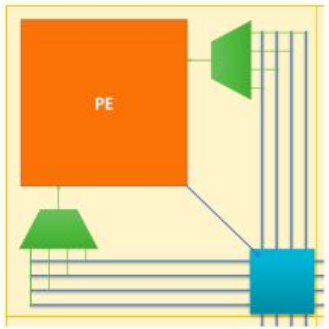
Virtual tapeout

Using Gemstone for Constructing Physical Flow



Using Gemstone for Constructing Physical Flow

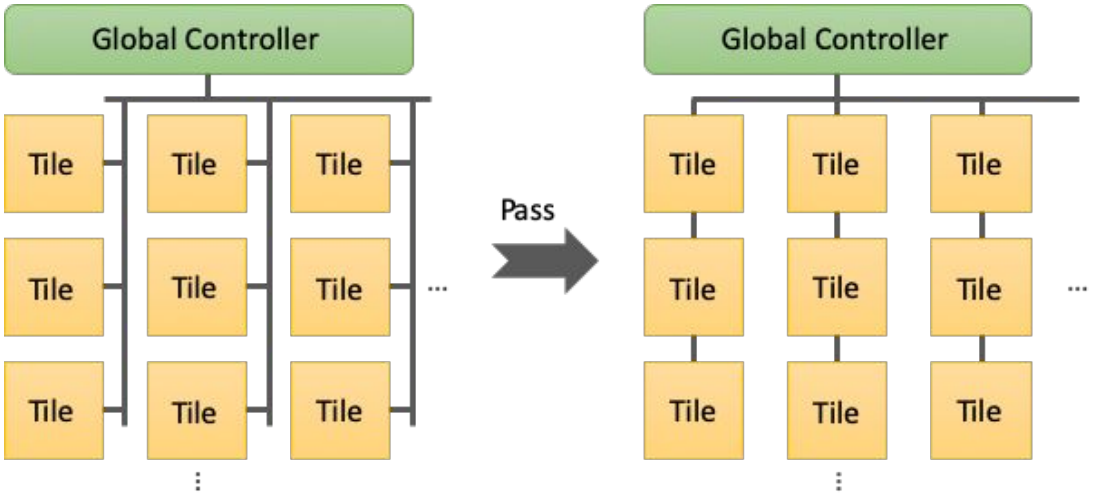
- Already using gemstone passes for
  - Adding configuration registers




```
class PE(Configurable):
    def __init__(self, params):
        # In reality these are coming from a peak spec.
        self.add_config("opcode", 5)
        self.add_config("reg_mode", 2)

        # Rest of the logic...



garnet = Garnet()
garnet.tile[(x, y)].core.config["opcode"].addr
```

Using Gemstone for Constructing Physical Flow

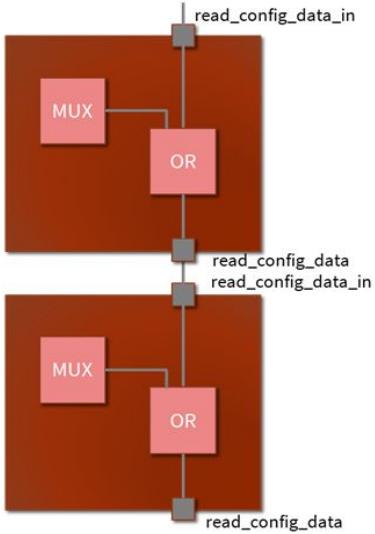
- Already using gemstone passes for
  - Adding pass through connections for river routing of global signals



River routing of global signals

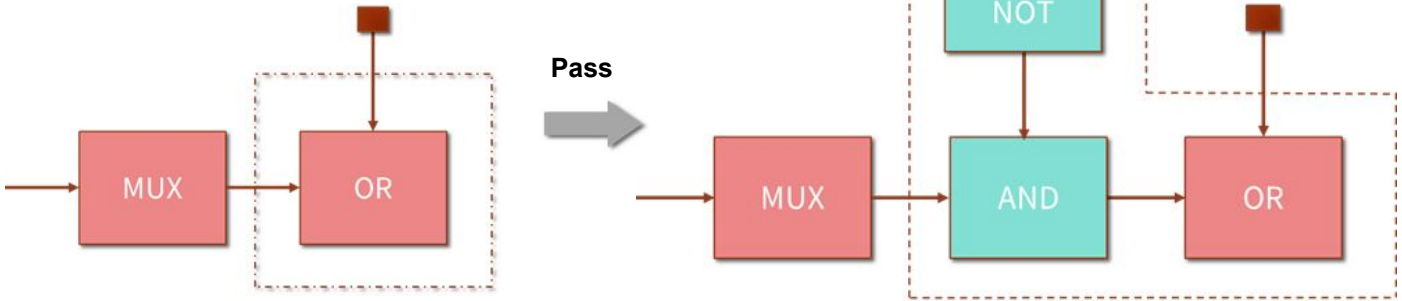
Using Gemstone for Constructing Physical Flow

- Already using gemstone passes for introducing power domains
  - Replacing muxes with AOI muxes for boundary protection
  - Adding configuration registers that control power switches
  - Keeping read_config_data ON when the bottom tiles are OFF



River routed global signals

```
When PD_CFG_REG= 0 (Tile ON)
  -   Normal mode of operation (OR logic)
When PD_CFG_REG= 1 (Tile OFF)
  -   Clamping operation
```

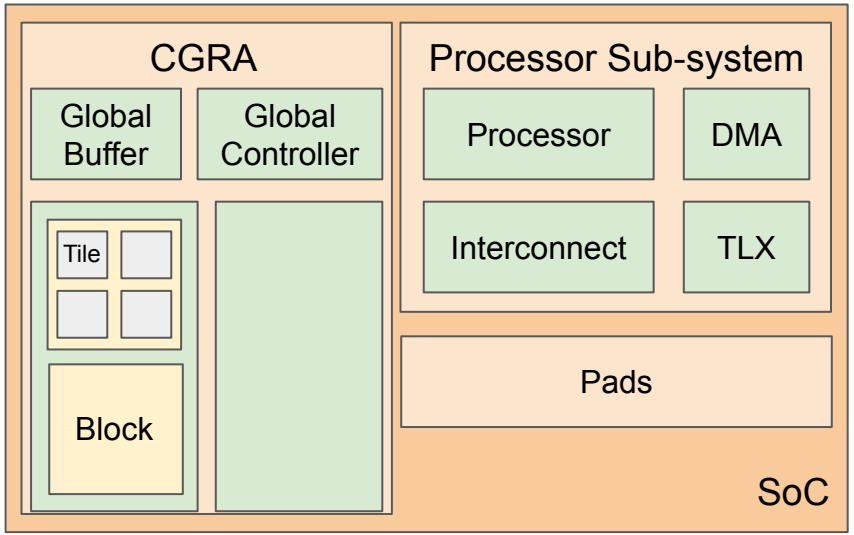


Opportunities for Passes

- We want to construct a hierarchical physical design flow with passes
  - Meaning that we can run an end-to-end flow on each block
  - Block designer knows best how to constrain the design
  - This will allow quicker assembly and easy debugging, but layout may be less dense

- Global passes need to determine
  - Optimal shape, size and position of blocks
  - Placement of pins
  - Retiming at interfaces
  - Buffer insertion

  Need a view of all objects at a level of hierarchy



Opportunities for Passes

- For each object, instead of a populating a parameterized monolithic place and route script, we will use passes to construct that script in a staged fashion

- Different passes generate different physical design attributes for an object
  - Timing constraints, e.g., gemstone has a special type for clocks - it can associate all signals with a clock domain. Passes can
    - Identify all clocks and constrain their periods
    - Check that signals cross clock domains safely
    - Generate false paths constraints between domains
    - Similar type-based rules for constraints ‘configuration’ type signals - false paths, data checks
  - Placement constraints on hard macros such as SRAMs (affects routability, timing)
  - Power and ground grids

Virtual Tapeout

- Continuous integration of physical flow
  - Leveraging our build kite infrastructure
  - Running all steps of the flow including final fill, DRC, LVS, STA and gate level simulation
- Requires passes that generate the place and route scripts from the design
  - So there are no manual updates to scripts for incremental design changes
- Has to be staged because Innovus doesn’t fail loudly
  - At every stage we have to identify indicators for a downstream failure, concerning values for them
  - Like check for hotspots after placement which can cause routing congestion
  - Could be automated with text processing of logs or getting indicator statistics at every stage from the tools