A brief history of rocket-chip

- UCB-BAR building RISC-V test chips
- Shared infrastructure to contain different student projects: Caches, cores, accelerators
- Decided to release publicly to help people use RISC-V
- Initially tried releases but eventually moved to open development
- First implementation of tilelink
  - Designed for on-chip interconnect, and cache-coherence
- Rolled its own parameter system
Issues with this version

• Global parameters need to be in sync
• Some parameters are derived and should be based on what the scala code will dynamically create
• This wasn’t supported so things were brittle
• Had to specify when bitwidths would change in TileLink
• Headache to maintain different top-level projects
  • Needed to understand all parameters of subprojects to integrate
  • Lots of code duplication
Diplomacy

- A way to negotiate parameters with explicit two-phase compilation
- LazyModules are the first stage
  - Create and connect different Node types to make “requests”
  - Other modules also make “requests” with their Nodes
  - ”Requests” are resolved when a LazyModule is realized with `.module`
- LazyModuleImps are approximately normal chisel modules
- Multiple sets of Nodes can co-exist
  - For example, TileLink Nodes and Interrupt Nodes
- A lot of Scala under the hood but the interface is designed to be useable without knowledge of them
Peripheral “Cake” Pattern

- Use traits to mix-in functionality at the top-level
- Avoid code duplication of “Top”s by allowing traits to create their own connections
  - Some traits supply access to system level buses and signals
  - Mix-ins grab those and wire themselves up
- Historically there were 3 traits needed (diplomatic, bundle, wiring)
- New chisel feature has eliminated the need for the bundle trait
- Offers different integration points for different kinds of peripherals
  - Memory bus, system bus, peripheral bus
Capital “P” Parameters

• Global map of "top-down" parameters
• Informing what the generator should build
  • E.g. how many cores, what in the cores, etc.
• Probably won’t go away completely but many of them have been and will continue to move to Diplomacy
• Recommended to use case class for your widgets parameters to group them and give sensible defaults
Modern Rocket-Chip

• Continuing to rapidly evolve
  • Improving usability and robustness
    • Generator approach means there will always be more configs to try
  • Refactoring often to alleviate new/old issues
• Many consumers and requirements
• Also supports accelerators and other configurations to customize
Testing Strategy

• Chisel tester for normal Modules
• Unit-test configurations
  • Fast build times, iron out
• PatternPushers
  • Test specific paths in directed fashion
• Fuzzer
  • Test random transactions
• Parameter checking
  • Do your unit-test nodes match integration nodes?
• Integration Tests
  • C code that peeks/pokes your SCRs
Unit tests

- Lets you debug simplest level of bugs
- Build your widget in isolation with whatever diplomatic connections you want
- Verify the basic functionality is working
  - Iterate quickly
- PatternPusher enables specific sequences of reads, writes, and results
- Fuzzer is generic random tilelink transactions
  - Only checking for validity/assertions
Integration

• Talk to your integrator
  • Especially if you have top-level ios or analog blocks
• Add your two mix-ins
• Set your Parameters to normal values
• Write C/ASM to exercise your widget
  • Ideally the cross product of widgets is small
Imitation is the sincerest form of flattery

• Examples exist in many places
  • Freechipsproject/rocket-chip, ucb-bar/testchipip, ucb-bar/project-template, sifive/sifive-blocks

• Widgets specifically often follow one of two models for input
  • Control registers that can be defined with RegisterRouters
  • TileLink Client/Manager for bandwidth

• Most widgets then have some special signals that are output to the outside world or other peripherals/widgets
Questions?