FPGA chip verification using UVM

Ravi Ram

Principal Verification Engineer

Altera Corp

Charles Zhang

Verification Architect

Paradigm Works

Outline

- Overview
  - Verilog based verification environment
  - Why UVM?
  - New UVM based verification environment
  - FPGA chip verification flow
- Some of the challenges and solutions
  - Generic programmable logic
  - Legacy code integration.
  - Programmable core & IO connection
  - VIP integration(external and internal)

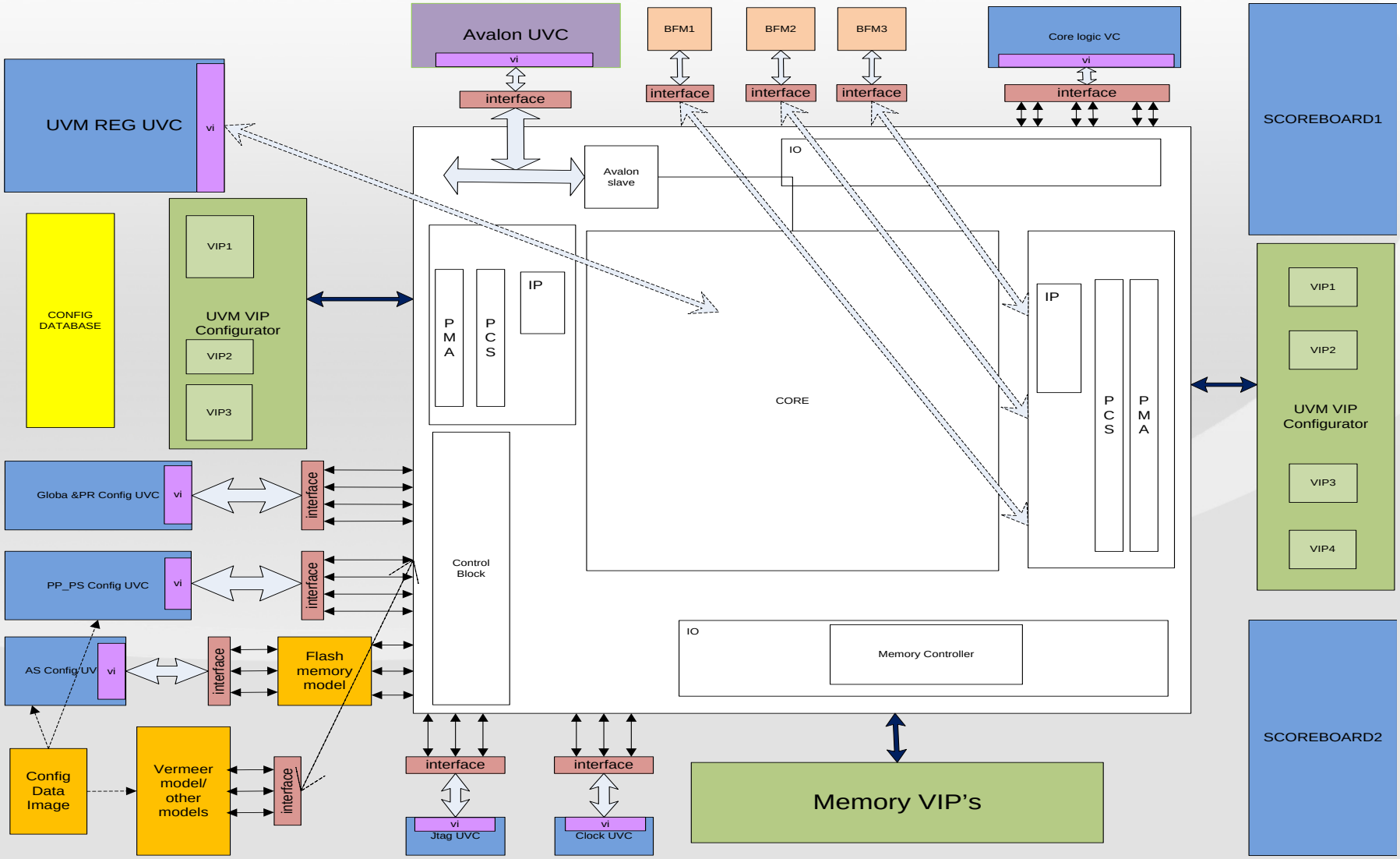
Verilog based Verification Env

- **Traditional Verilog based verification environment**
- **Multiple test benches for multiple modes of operation**
  - PP, PS, SPI, USERMODE, etc.
- **Recompilation for each test**
- **No object oriented programming (reuse = copy and change)**
- **Maintainability and scalability are poor (large number of tests, etc.)**
- **Easier for designer to understand and modify**

Why UVM?

- Supported and released by Accellera
- Supported by all major EDA vendors
- Object orient programming
- Reusability
- Well defined base class library
- Industry standard makes integration of third party or home grown VIP easier
- Good online documentation + UVM forums etc
- Little bit harder for designer to understand

UVM based Verification Env Overview



UVM-based verification Env overview

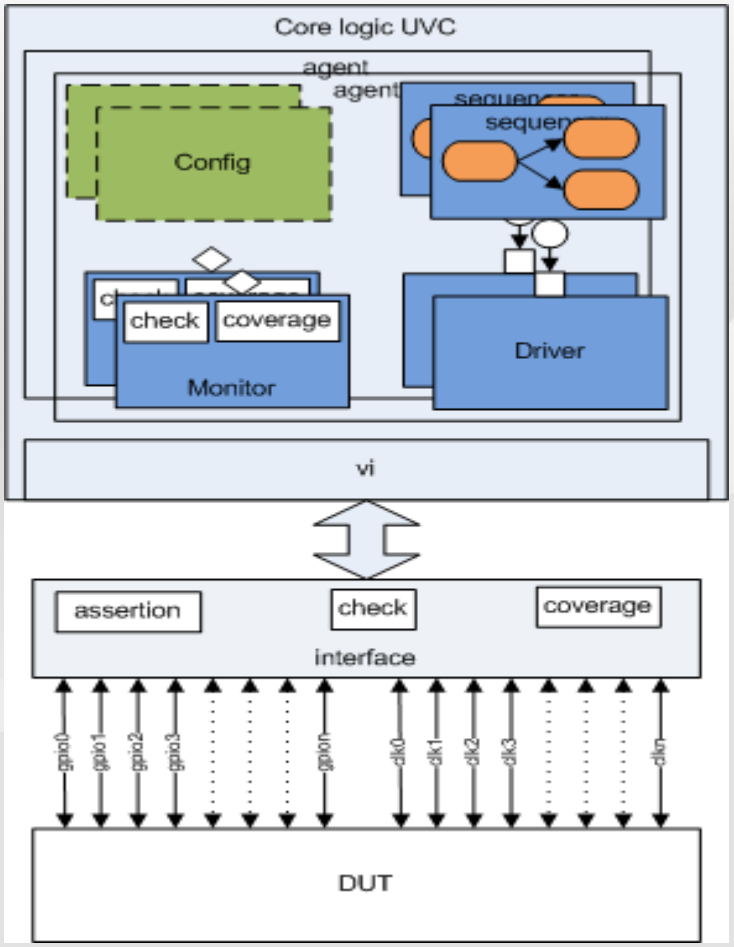
- Architected from scratch
- One environment supports multiple operating mode
  - PP, PS, SPI, USERMODE, etc.
- Significantly reduced number of tests by inheritance, configuration setting, etc
  - The current UVM based tests is about 1/3 of the tests of Verilog based ENV
- Simulation performance improved by compile once and run multiple tests
- Improved compile, run and regression flow
  - With UVM, cmd line processor is built-in and free

FPGA Verification Flow

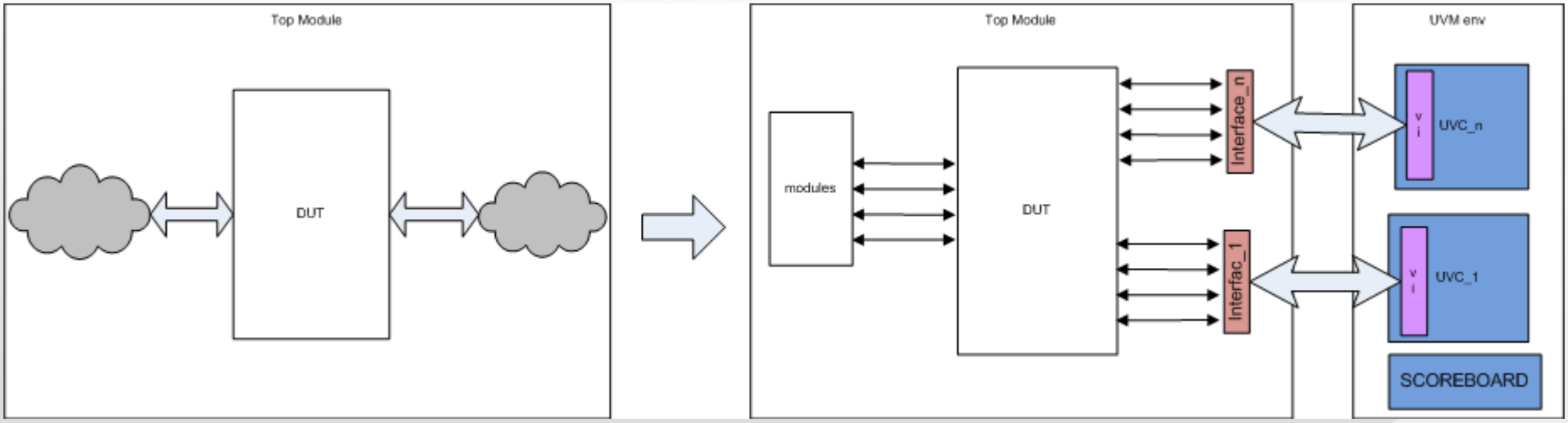
- Configuration (Programming the FPGA).
  - Support multiple programming interfaces
  - Data compression and encryption
  - Front door and back door loading configuration
  - Verification goal: make sure the programmed image matches the expected image
- User Mode (Running programmed user logic)
  - Tests include testing all core logic blocks and all the IO systems
  - Considerable effort is on creating configurable verification environment
  - Verification goal: verify all the core blocks and I/O systems to be functioning and connected properly

Generic programmable logic

- Programmable nature of FPGA calls for programmable verification environment
- Core logic interface UVC is a highly programmable verification component.
  - Allows user to decide on which pins to drive using UVM configuration
  - The monitor extended by user to implement any checking mechanism using UVM factory override.
  - Test based on sequences and transactions without worry about pin connection and toggling.
  - Compile once and run all tests.
- Used by the software group to verify real customer design.



Legacy code integration



- Still need Verilog based verification environment coexist with UVM verification environment
- Interface file used as bridge between UVM verification environment and verilog based verification environment
- Interfaces bound to its physical interface signals
- Virtual interface in UVC set by getting the instance from resource database
- Assertions implemented in interface binds to module or physical interface signals

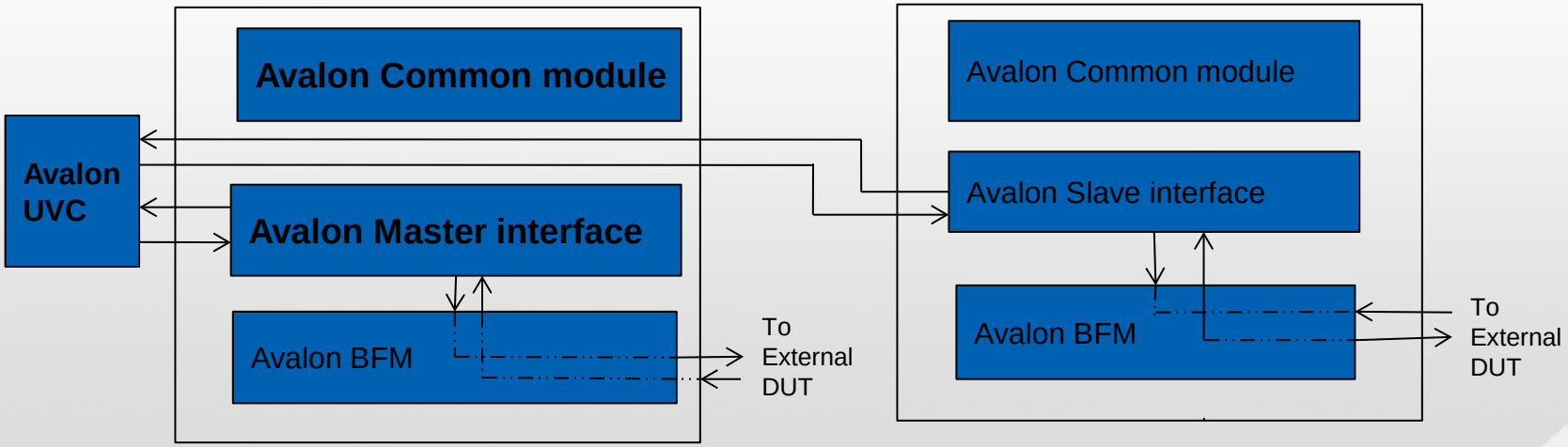
Programmable core & IO connection

- FPGA core is programmable
- All hard IP is configurable
- Lots of different interfaces and VIPs
- Register access from reg UVC to configure FPGA
  - Thousands of configurations in FPGA. UVM Reg model is already > 20G for handling 30 to 40% of the FPGA configurations. So this is not scalable and not practical to use
- Hundreds of configurable registers which UVM reg based testing cannot handle
  - Use home grown resource allocator plus configuration settings
- Register access from reg UVC to configure FPGA
- Seamless integration of resource allocator(internal tool) with internal developed tools for unidirectional and bidirectional connections

VIP integration

- **Lots of VIPs to address hard IP in FPGA(1G/10G…, PCIe plus other serial protocols, Altera Avalon VIP, different memory VIP for different memory protocols)**
- **Flexibility to configure and select VIPs in UVM test**
- **Use constraints to select the connections and VIPs**
- **Use on the fly point-to-point connections to connect VIP to the fabric**
  - Turn off unused VIPs
- **Same environment for integrating different vendor VIPs**
- **Environment setup for proliferation products for same FPGA family**
- **VIP interface easily portable to future FPGA families**

Avalon VIP Integration



- Integrate Avalon BFM in UVM environment
- Use of the existing bfm with a wrapper on top to make it a UVC
- VIP developed internally in Altera and is made available for use by all groups
- The configuration object generated for each instance of the VIP with a unique hdl Path which has a reference of the interface.
- The user provides the parameters for the VIP and the hdl Path in his test-bench hierarchy

Summary

- Altera's first verification project adopting UVM
- Addressed critical challenges
  - Programmable user logic and io
  - Explosive configuration spaces, etc.
- Adopted pragmatic view of the methodology
  - Re-architected the whole environment using UVM
  - Reused and integrated both internal and external VIPs
- UVM provides ideal way to create configurable, reusable verification components and environment

Q & A

Thank You!

Contributor: Manish Mahajan