Tutorial 2A. Practical Exercise: Get Your FPGA Application Up and Running

December 10, 2008
Introductions

Lee Goldberg
Daniel Platzker
Steve Knapp
Dave Orecchio
Sanjay Thatte
Andy Stevens
Dan Isaacs
Order of Business

- **Four short presentations** (15 minutes each)
  - Design Tools and Design Portability
  - Test Tools and Equipment
  - Verification
  - From System Specification to FPGA Implementation

- **Breakout sessions on key challenges**
  - Design Methods
  - System Interface
  - Verification

- **Summary and panel session**
Design Tools/Design Portability
Tips and Tricks for Vendor Neutral FPGA Development

Daniel Platzker
FPGA Product Line Director
Mentor Graphics Corp.
Why Vendor Neutrality: Device & Tools

Important to retaining flexibility to switch vendors

• Existing vendors leapfrog each other on capabilities/cost
• New vendors offer unique capabilities, (Achronix, eAsic)
• Retain price leverage with vendors

Single tool flow less expensive

• Reduce maintenance and training
• Objective device selection
Users Rate Importance of Vendor Independence

All attributes rate above average!

- One synthesis tool for all FPGAs
- Single constraints for all FPGAs
- Re-use IP regardless of FPGA
- One tool to help select FPGA

Source: Image & Market Measurement, January 2006
What is Required for Vendor Neutral Development

1. Vendors neutral tools

2. Vendor neutral methodologies
   - Industry standards
   - Vendor neutral code
   - Wrappers

3. Minimal use of vendor dependent IP

4. Discipline
1. Vendor Neutral Tools

All tools in the flow should support multiple vendors:

- ESL
- Design Creation
- Synthesis
- Equivalence checking

The Exception

- Place & Route
2. Vendor Neutral Methodologies

- Higher levels of abstraction
  - C++, Mathworks, SystemC
- Industry standards
  - SDC, LRM compliance
- HDL coding style
  - Inference instead of instantiation
- Wrappers for vendor dependent IP
  - DCMs, PLLs

```vhdl
always @(posedge clk)
begin
  mult0_result <= A * B;
  PCOUT <= mult0_result + PCIN;
end
```

Technology
Independent Inference
3. Minimize Vendor Dependent IP

- **Best:**
  - vendor independent IP
  - In-house or 3rd party IP vendors

- **Acceptable:**
  - Everything with standard interface that can be swapped
  - Wrappers needed for some IP (e.g., DCM, PLL)

- **Challenge:**
  - Vendor CPUs (Software, peripherals)
  - Consider portable CPUs, e.g., ARM, Gaisler, Tensilica
4. Discipline

Management needs to endorse vendor neutral development tools and methodologies and stick to it.
Summary

Vendor neutrality = ability to select the best FPGA for your next project, regardless of silicon vendor

- **Why?**
  - Existing vendors leapfrog each other on capabilities & cost
  - New vendors offer unique capabilities e.g., Achronix, eAsic
  - Allows for price leverage with vendors
  - Single tool flow less expensive (vs. flow per vendor)

- **What is necessary?**
  - Vendor neutral tools, methodologies, IP, discipline
Test Tools and Equipment

Steve Knapp
Principal Engineer
steve.knapp@prevailing-technology.com
... and Everything Else

- It’s obvious that you need FPGA devices and related design software
- But what else is required to be successful?
  - Development boards
  - In-system programming, download, debug cables
  - Device programmers (for some devices)
  - Logic analyzers, signal generators, oscilloscopes
  - In-system debugging tools
  - Test sockets
- The mundane minutiae of success
Lab Time is Expensive!

- Find and fix bugs before you enter the lab!
- Simulate first
  - Create good quality simulation models and testbenches
  - Leverage hardware in the loop if possible
    - Improved accuracy (sometimes)
    - Shorter run times
- The difference between theory and practice
  - Simulation will not find all the bugs
  - But, it is an important and necessary first step
Development Boards

- Build or buy?
- Development boards speed development
- Generic or application specific
- Cost from $40 to $10,000+
  - Inexpensive, **expendable** board, one per engineer
  - More expensive, application-specific board for team
In-System Download/Debug Cables

- Supplied by FPGA vendor, although 3rd party solutions available
  - Sometimes built into development board
- Connects FPGA or associated configuration memory to PC or workstation
  - USB / Ethernet / parallel port
- Provides direct downloading, programming, and debugging capabilities
- A must have!
Device Programmers

- Required to program nonvolatile memory
  - Some FPGAs have internal nonvolatile memory
  - Others use a companion memory device
- Prototyping (desktop) vs. production programmer
- Value-added programming services offered by some vendors, distributors
- Don’t forget the required socket adapters
Test Equipment

- What you need depends on the complexity of application?
- Logic analyzers
- Signal generators
- Oscilloscopes
  - Differential probes
  - Jitter measurement
- Bit error rate tester (BERT)
  - High-speed serial applications
In-System Debugging Tools

- Visibility is key to successful debugging
- Most horrors are buried deep inside FPGA
- Some vendors offer ability to view internals
  - Xilinx ChipScope
  - Altera SignalTap II
  - Altium, Synopsys,
  - National Instruments
- Debugging tools for processor-based design
Test Sockets

- Many new FPGAs use advanced, high pin-count, surface-mount packages
- Many new FPGAs are high-performance with associated signal integrity concerns
- Sockets can be challenging and difficult to find
  - Embedded Technology
  - Ironwood Electronics
  - Molex
The Key to Success?

- FPGA design is much more than just devices and design software
- These extra items do not guarantee success
- But, will make success come easier
- Skimping on this equipment will cost you far more in the long run
- Advanced capabilities like high-speed serial I/O and embedded processors require specialized tools
FPGA Summit, Tutorial T2A, Wednesday December 10, 2008

FPGA Verification

Dave Orecchio, President and CEO
Each year, FPGAs address more markets
But designs are becoming too complex to test and debug

Source: FPGA Companies
10. How critical are the following issues and challenges with your current FPGA projects? (Select one best response for each issue / challenge)
10a. How critical will these issues and challenges become – as process geometries continue to shrink?
Where do errors originate?

Electronic Design Software World

- Specification
- Design Errors
- IP Blocks
- IP Bugs
- Tool Bugs
- Design Verification
- Synthesis – P&R
- Hardware Design

Hardware World

- Hardware Verification & Debug
- Hardware/Software Debug
- Product

Specification errors
Where are errors detected?

Electronic Design Software World

- Slower execution
- Can’t see tool or IP bugs
+ Excellent visibility
+ Easy to debug & fix

Hardware World

+ Faster execution
- Poor visibility
- Hard to debug & fix
- Long cycle times per iteration

GateRocket
Electronic Design Software World

Functional Verification = Testbench + Engine(s)
Typical Test Bench Approaches

- **HDL-based**
  - Directed tests
  - Randomized tests
- **HVL-based (SystemVerilog, e, Vera, SystemC)**
  - Constrained, randomized
  - Class-based/object oriented
- **Embedded software-based**
Verification Engine Tradeoffs

Ease of debugging
- Visibility & controllability
- Fast design iteration

High Speed
- Long setup/change time
- Low visibility and controllability

100% software

100% hardware

- HDL Simulation
- Hardware Acceleration
- Hardware Emulation
- Prototype Boards
HDL Simulation

- Typical application: block-level verification
  - Fast design change turnaround
  - Full visibility/controllability

- Testbench methodology:
  - From: simple directed + randomized tests
  - To: complex object oriented constrained-random (HVL)

- Challenges:
  - Chip-level verification may require compute farm
  - Long “unbreakable” test sequences can take days/weeks
  - Don’t see chip-level or device specific IP issues
Hardware Acceleration

- Typical application: block-chip regression test
  - 1-2 orders of magnitude faster than simulation
- Testbench methodology:
  - Typically directed tests with simple randomization
- Challenges:
  - Capital cost
  - Setup time
  - Matching results with simulation
  - Specific hardware or software flow makes it impossible to verify designs with FPGA specific IP blocks
Hardware Emulation

- Typical application: chip/system regression test
  - 3-4 orders of magnitude faster than simulation
  - Typically uses different mapping than target FPGA
- Testbench methodology:
  - Embedded code
- Challenges:
  - Capital cost
  - Setup time
  - Matching results with simulation and target FPGA
  - Debugging
  - Flow makes it impossible to verify designs with FPGA specific IP blocks
New: Silicon Accurate, Device Native Verification

Uses the native FPGA device as the accelerator
Device Native Verification

- Typical application: block-system regression test
  - Order of magnitude speed-up over simulation
  - Full debug visibility
  - Silicon-accurate results
- Testbench methodology:
  - Same as HDL simulation
- Challenges:
  - Inefficient testbench can degrade acceleration
Conclusion

- Thorough functional verification prior to system integration is a must for complex FPGAs
- Simulation coupled with hardware assisted verification offers the best of all worlds
  - Performance/coverage, debugging
- Device-native verification offers a promising, silicon accurate approach
Tutorial T3B 2:40 to 5:00 PM Today

- Joe Rodriguez, Prod Mktg Mgr, Mentor Graphics
  - Presentation: Recalculating the Road to Successful FPGA Verification
- Chris Schalick, CTO, GateRocket
  - Presentation: A Case for Hardware-Assisted Verification
- James Smith, Dir EDA Vendor Relations, Altera
  - Presentation: FPGAs Increasing Role in Development
- Judith Smith, Prod Mgr, Agilent Technologies
  - Presentation: FPGA-Based DDR Memory Controller Validation
- Carsten Hoffman, Emulation Platforms and Dan Isaacs, Dir Adv Prod Mktg, Xilinx
  - Presentation: Platform FPGA Automated Regression Test Methodology
System Design to FPGA Implementation

FPGA Summit
December 2008
Agenda

- Scope of presentation
  - Different methodologies possible
  - Common theme - Tackling FPGA design complexity

- FPGA design challenges

- Possible design flow solution
  - System Design
  - RTL Design
  - Physical Implementation
  - Verification
FPGA Design Challenges

- Challenges
  - Business driven
    - Reduce cost
    - Meet performance goals
    - Reduce time to market
  - Technology driven
    - Device size and complexity
    - Design size and complexity
    - Certifiable design process

- Solution - ASIC style design methodology
  - Raise level of abstraction
  - Reuse designs
  - Maintain vendor independence
  - Use advanced verification tools
FPGA Design Flow

Planning
- Requirements Capture
- Conceptual Design
- Create RTL

Synthesize Design

Place, Route
- Program or Mfg Device

Verify RTL Design

Verify Gate-Level Design

Test HW Device

Trace Requirements Throughout The Flow
Impact Of Change

Impact

90%

30%

10%

ESL, RTL, Gate, Backend

SW / Macro Architecture

Micro Architecture

Logic

Physical

SW
Raise Level Of Abstraction

Scalable Transaction Level Modeling

- Single C/C++ source for synthesis, verification and analysis
- Automated timing annotation from HLS scheduler
- Fast System simulation at TLM speed [x1,000 and up]
Raise Level Of Abstraction
Transaction Level Modeling Solutions

Modeling
- Functional
- RTL
- Automate Modeling

Assembly
- ESL/RTL Integration

Validation
- HW/SW Validation

Analysis
- Optimize System Architecture
Raise Level Of Abstraction

HW/SW Debug & Analysis Platform

- Key Features
  - Powerful HW & C/C++ views
  - TLM Tracing
  - Powerful Debug – Process Control
  - Timed / Untimed simulation
  - Timing / Power Analysis

- Key Benefits
  - Reducing validation cycles
  - Bridging SW & HW Design
  - System-level Exploration
  - Standard Based – OSCI/gdb/gcc
Raise Level Of Abstraction
ESL Synthesis

- Use flows that are certified
- ESL Synthesis
  - Synthesizes standard ANSI C++
  - Technology aware scheduling
  - High quality RTL code generation
- ESL output optimized for FPGA Synthesis
- FPGA Synthesis integrated with ESL tool

![Diagram showing the process flow of ESL Synthesis and FPGA Synthesis integration.](Image)
Raise Level Of Abstraction
Better QoR with ESL-FPGA synthesis flow

- **Technology aware C++ synthesis**
  - Allocation, Scheduling based on timing and area of library elements (Library elements: logic gates, arithmetic operators, memories)
  - Timing and area information extracted from FPGA synthesis tool (Library characterization process)

- **ESL tool outputs FPGA synthesis tool ready RTL**

- **FPGA synthesis tool includes ESL tool libraries**
  - Latest FPGA timing data available to ESL tool
Raise Level Of Abstraction

Close integration - ESL and FPGA Synthesis

- FPGA Synthesis invoked from ESL with project settings being carried over

- Extensive one-step cross probing from FPGA Synthesis and Place & Route result files to ESL tool

- FPGA Synthesis is tested in ESL tool regression framework
Reuse Designs

HDL Design Environment

- Save time by analyzing & correcting design integrity
- Ensure the best reuse decision by measuring code quality
- Accelerate design understanding of structure & behavior
- Extract, package and manage IP
Maintain Vendor Independence

FPGA Synthesis

- **Efficient Design Creation**
  - SystemVerilog & standards support, ESL integration

- **Physical Synthesis**
  - Broad device support
  - Superior results for all vendors

- **Incremental Synthesis**
  - Support for incremental design flow

- **Graphical Resource Analysis & Control**
  - Allocate resources to optimize for performance or area

- Improve QOR
- Reduce cost
- Speed up time to market
Maintain Vendor Independence
FPGA Synthesis

- **Analysis & Debug**
  - Missing Constraint report
  - Clock Domain Crossing report
  - Comprehensive messaging/warnings
  - Cross-probing between HDL & Schematics
  - Critical-path viewing for timing analysis
  - What if analysis without re-synthesis
  - Graphical resource analysis and control

- **Technical Support**
  - Support complete flow with proper ownership
Maintain Vendor Independence

FPGA Synthesis

- **Safe Operation**
  - Safe FSM encoding to avoid single-event upset
  - Support for Redundancy
  - Safer synthesis with selective optimization

- **Verifiable & Reproducible**
  - Good integration & support with verification tools
  - Deterministic netlist generation

- **Synthesis Artifacts**
  - Extensive reporting for documentation requirements

Improve Design Safety
Use Advanced verification tools

Advanced Functional Verification

**Inputs**
- Design
- Testbench
- Assertions
- Verification Plan
- Power files

**Languages**
- VHDL
- Verilog
- PSL
- SystemC
- SystemVerilog
- C/C++

**Outputs**
- Waveforms
- Functional Coverage
- Code Coverage
- Toggle Coverage
- Assertion Coverage

**Analysis**
- Hardware debugger
- UCDB
- Assertion debugger
- C debugger
- Verification management
Use Advanced verification tools

Formal Verification

- 100s times faster than gate level simulation
- 100% coverage provided
- Resolves netlist uncertainty
- A complete gate-level test bench is time consuming and large
- Place & Route takes a day
- Running FPGA live does **not** test the corners
Use Advanced verification tools

Manage Requirements in Design Flow

- Control & predict project schedules
- Trace requirements through HW design process
- Clearly communicate via visualization & intuitive reports
- Manage impact of requirement changes
- Meet safety critical & DO-254 certification

Requirements Capture & Tracing

Report & Document

Change Management

Trace Through
Authoring, Coding & Testing

Link Automated Reports, Specs, Design & Results

Impact Analysis to Control Schedules
Physical Implementation

- **Input data**
  - Synthesized design netlist
  - Design constraints

- **Improving performance**
  - Different implementation options
  - Design exploration tools
  - Interactive physical design
  - Different synthesis options
  - Modify RTL

- **Debug**
  - Insert debug logic post-synthesis

- **Program the device**
  - Create bit stream
Summary

- FPGA design challenges are growing
- Design process needs to be certifiable

- Solution is to use ASIC style design methodology
- To enhance effectiveness
  - Raise level of abstraction
  - Reuse designs
  - Maintain vendor independence
  - Use advanced verification tools
Breakout Sessions

- Break into three groups
- Take next 20-30 minutes to discuss topics
  - Design Methods — Daniel Platzker
  - Systems Interface — Andy Stevens
  - Verification — Dave Orecchio
- Report back as part of the panel discussion
Design Tools & Portability: Breakout Questions

- How important is vendor independence (VI) to designers vs. management?
- What are the advantages/disadvantages of using vendor vs. EDA tools?
- When do you write VI code (inferable) vs. instantiating tech cells?
- Do you look at different vendors for each project? Why not?
  - No time?
  - Strong existing vendor relationship?
  - Silicon requirements met by single vendor?
  - IP?
- Using Vendor IP? Why?
  - Because of cost?
  - Happy with the vendor support?

Which IP using the most? Processors? SERDES?
Panel Session

- Reports back from breakout sessions
  - Design Methods — Daniel Platzker
  - Systems Interface — Andy Stevens
  - Verification — Dave Orecchio
Thank you for attending!

- Please fill out the session evaluations and return them up front
- Key note address
- Lunch time!