Verification I

Sungho Kang

Yonsei University
Outline

- Introduction
- Definition
- Hardware Acceleration
- Emulation
Complexity Trends

- **Rapidly Growing Design Size**
  - Doubling of million-gate designs
  - 50% reduction in designs under 500K gates
  - 3X reduction in designs under 100K gates

- **Shrinking Process Geometries**
  - Nearly 10X reduction in .5 micron designs
  - Most designs going to .35 micron or below

- **Architectural Complexity**
  - Before: simple instruction pipelines, single functional units, simple stalls/holds, simple caches/TLBs, protocols at the pins
  - Now: deep instruction pipelines, super-scalar design, multiple (pipelined) functional units, instruction re-circulate, speculative execution, complex stalls/holds, complex protocols on chip and at the pins, integration of external IP
Verification is the Biggest Problem

- Verification groups growing faster than design groups
- Verification tools are largest budget item for many groups
- Expenditures are growing
Informal Verification

- Simulation
  - Compare against an executable version of the specification, also known as THE GOLDEN MODEL
  - Simulate in software
  - Simulate in hardware

- Test cases
  - Hard-written by the designers
  - Randomly generated test vectors
Trends: Verification Problem

- Number of test vectors proportional to design complexity
  - Simulation cannot guarantee correctness
  - Confidence based on proportion of design space explored
- Hardware, software systems are becoming increasingly complex
  - Number of basic components growing exponentially
  - Designs are increasingly aggressive
Why Specify

• Additional documentation of system’s interface
• More abstract description of system design
• To perform some formal analysis of the system
Why Specify

Abstraction Level 5
Abstraction Level 4
Abstraction Level 3
Abstraction Level 2
Abstraction Level 1

Top-down Design
Bottom-up Design
Verification

- Hardware Acceleration ✓
- Emulation ✓
- Co-Verification
- Formal Verification
Hardware Acceleration

Sungho Kang
Yonsei University
Outline

• Introduction
• Boeing
• TEGAS
• Yorktown Simulation Engine
• Logic Simulation Machine
• HAL
• ZYCAD
• AAP-1
• Reconfigurable
Why Simulation Engine

- Speed up difficulty in software simulation
- Parallel Processing
  - Multi-processing
  - Pipelining
  - Array Processing
- Hardware Implementation

- Simulation Engine / Hardware Accelerator
  - Compiled
  - Event Driven
  - Multi-Processor
  - Array
## Simulation Engine Performance

<table>
<thead>
<tr>
<th></th>
<th>Architecture</th>
<th>Maximum Evaluation Units</th>
<th>Simulation Algorithm</th>
<th>Maximum Gates</th>
<th>Announced Speed</th>
</tr>
</thead>
<tbody>
<tr>
<td>YSE (IBM)</td>
<td>Multi-Processor Pipelining</td>
<td>256</td>
<td>Compile</td>
<td>1M (4 input / 1 output gate)</td>
<td>2000M (gates/sec)</td>
</tr>
<tr>
<td>HAL (NEC)</td>
<td>Multi-Processor Pipelining</td>
<td>31</td>
<td>Level Controlled Event Driven</td>
<td>1.5M</td>
<td>300M (gates/sec)</td>
</tr>
<tr>
<td>LE (ZYCAD)</td>
<td>Multi-Processor Pipelining</td>
<td>16</td>
<td>Event Driven</td>
<td>1.6M (2 input / 1 output gate)</td>
<td>16M (gates/sec)</td>
</tr>
</tbody>
</table>
Simulation Engine Classification

**Introduction**

Simulation Engine Classification

- **Array Processor Architecture**
  - Simulator on AAPI (NTT)
- **Multi-Processor Architecture**
  - YSE (IBM)
  - HAL (NEC)
- **Hardware Event Driven Architecture**
  - LE (ZYCAD)
  - LSM (Bell Lab.)

**Actual Circuit**

- Hardware Mock-up
- Software simulator (Compile)
- Software simulator (Event driven)

**Number of Parallel Processing**

- 655366 processors
- 256 processors
- 31 processors
- 16 processors
- 5 stage pipe
- 1 processor
- 5 stage pipe

**Number of Parallel Processing**
TEGAS Accelerator
TEGAS Accelerator

- Functional Level Block Diagram
TEGAS Accelerator

- Accelerator Update Processor

Diagram showing connections and components:
- LIFO Address and Access Control
- Master Controller
- Simulation Processing Memory Access Controller
- Support Bus
- To / From Simulation Processing Memory
- From Update Processor LIFO Memory
- From Host Interface Processor
- To/From result Buffer Memory
- To/From Control and Statistics Processor
- To/From Fault List Processor
- From Evaluation Processor
- Descriptor Address Pipe
- From Host Interface Processor
- To/From Host Interface Processor
- Support Bus
TEGAS Accelerator

- Accelerator Evaluation Processor

Diagram showing various components and their connections, such as Behavior Processor Instruction Memory, Behavioral Language Processor, Behavioral Buffer, Structural Buffer, and others.
TEGAS Accelerator

- Accelerator Time Queue Processor

Simulation Processing Memory Address Bus

Support Bus

From Activity Search Processor

From Control and Statistic Processor

To/From Evaluation Processor

From Control and Statistic Processor

To/From Evaluation Processor

To/From Time Queue Memory

Time Queue Memory Availability Logic
Yorktown Simulation Engine

- Compiled

Diagram:
- 256 x 256 Switch
- Logic Proc 0, Logic Proc 1, Logic Proc 2, ....... Logic Proc 256
- Array Simulator
- Host
- Control Proc.
- Bus Control
Yorktown Simulation Engine

- Partitioning of 256 PUs
  - Each PU simulates a subcircuit consisting of up to 4k gates
  - Specialized PU for RAMs and ROMs
- All PUs are synchronized by a common clocks
- PU can evaluate a gate during every clock cycle
- Partitioning
  - Minimize the waiting time
- Control processor : host to YSE
Yorktown Simulation Engine

- Logic Processor

![Diagram of Logic Processor]

- Program Counter
- Fetch Address
  - Instruction Memory (1024 x 80)
  - Function Memory
- Operands 1-5
- Function
- Logic Unit
- Delay Value Memory 1024 x16
- DATA 2048 x 2 x 5
  - Signal Data
- Fetch Addr
  - Store Addr
- Fetch Addr

Yorktown Simulation Engine

- Logic Processor
  - PC provide the index to the next gate to be evaluated (compiled)
  - Signal values (0, 1, X, Z) are stored in data memory
  - Up to 4 inputs for each gate

- Generalized DeMorgan Code (GDM)
- 16 functions of 4 valued variables
- Evaluation is done in zoom table
Yorktown Simulation Engine

- **Function Unit**
  - Concatenating gate type with input values by GDM code

```
input 1
  value
  GDM
  GDM1 code
  Function memory
  GDM
  output
  value
input 4
  value
  GDM
  GDM4 code
  GDM5 code
  type
```

- **GDM**
- **Function Memory**
- **GDM1, GDM4, GDM5 Code Type**
## Event Driven Logic Simulation Tasks

<table>
<thead>
<tr>
<th>Processor</th>
<th>Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>CURRENT EVENT PROC</td>
<td>1) retrieve and distribute event</td>
</tr>
<tr>
<td></td>
<td>2) record event</td>
</tr>
<tr>
<td></td>
<td>3) oscillation check</td>
</tr>
<tr>
<td>MODEL ACCESSING UNIT</td>
<td>1) determine first fanout</td>
</tr>
<tr>
<td></td>
<td>2) determine next fanout</td>
</tr>
<tr>
<td>SIMPLE CONF PROC</td>
<td>1) update source configuration</td>
</tr>
<tr>
<td></td>
<td>2) update and transmit fanout configuration</td>
</tr>
<tr>
<td>EVAL</td>
<td>1) evaluate</td>
</tr>
<tr>
<td></td>
<td>2) check for repeated evaluations</td>
</tr>
<tr>
<td>SCHED</td>
<td>1) schedule</td>
</tr>
<tr>
<td></td>
<td>2) timing analysis</td>
</tr>
<tr>
<td>EVENT LIST MANAGER</td>
<td>insert in event list</td>
</tr>
</tbody>
</table>
Logic Simulation Engine

FROM STIMULUS FILE

EVENT LIST MANAGER

CURRENT EVENT PROC

MODEL ACCESSING UNIT

TO RESULTS FILE

SCHED

EVENT LIST MEM

UPDATES AND EVALUATIONS
Logic Simulation Engine

- Pipeline - uniformly distribute the work load
- How to solve bottleneck of special processors or FIFO buffers
- Element evaluation is indicated as soon as one of its inputs change
- Most time consuming
  - Fanout determination
  - Gate evaluation
- Function evaluations are executed concurrently by parallel processors
- Block Level Hardware Logic Simulator
• Event search
• Signal propagation
• System clock change
• Executes a level controlled event driven mechanism
• Simulation model consists of logic blocks
• 24 bit message packets solve conflicts when more than one access
ZYCAD Logic Evaluator

- Event driven
- Up to 16 processors
- Event processor - 5 stage pipeline
  - Get fanout list and determine delay
  - Read fanout and update output states
  - Update input states and determine gate type
  - Evaluate gate model
  - Schedule new event
- 3 input 1 output gate only
- Event stack - reduce overhead
- Future event scheduler - timing wheel
AAP-1

- Processing Element

8 NEIGHBOR PEs

MUX-ID1
MUX-RUT
REG-RS
MUX-OD

DO1
DTU1

PE LEVEL BYPASS

MUX-ID2
REG-10

MUX-ALDS
MUX-CRY
REG-C

DO2
DTU2
CO
RALU

'1'
CiU
CiR

RAM-B (32WX1b)
LAT-A

LAT-B
LAT-S

RAM-B (64X1b)

8 NEIGHBOR PEs

UPPER AND LOWER PEs

RF- A

RF- B
Reconfigurable Array Architecture

- Configuration
Mapping

• Mapping onto PE Array

(a) Example Circuit

(b) Mapping onto PE Array
• Expanded Circuit Mapping

(a) Expanded Example Circuit

(b) Mapping onto PE Array
Array Reconfiguration

A Circuit in a Netlist (G=4)

A Circuit in a Netlist (G=2)

(a) Two Levels are mapped onto the PE Array (P >= 2G)

(b) Four Levels are mapped onto the PE Array (P >= 4G)
PE Array Reconfiguration

- Folding

(a) P [PEs/level] with 1 level
(b) P/2 [PEs/level] with 2 levels
(c) P/4 [PEs/level] and 4 levels
Node Descriptor Memory

Reconfigurable

CS&RSOC

YONSEI UNIVERSITY
PE Cell Block

Reconfigurable PE Cell Block

Shifted Node Descriptor Memory

Switch Configuration
Gate Function
Flag
Output
Fanins in Good and Faulty Circuit

W Input
Fault Propagation Inputs

N-S I/O

I/O Switch

BEU
Output Latch and Comparator

Fault Handler

E Output
Fault Propagation Outputs
First Expansion

Reconfigurable
Expansion of XOR and XNOR

(a) XOR Gate Expansion

(b) XNOR Gate Expansion
Possible Expansion of 4 Input AND

Reconfigurable
Second Expansion

Reconfigurable
Fault Simulation Algorithm

make a fault list attached to each element for all patterns
perform good simulation
store the results of good simulation for all levels
insert the first fault in the fault list
simulate the fault
compare the value
if the value is different from that of good simulation
    propagate the fault to the next level
else
    drop the fault
insert the first fault in the fault list
simulate the fault
compare the value
if the value is different from that of good simulation
    propagate the fault to the next level
else
    drop the fault
end
end
Logic Emulation

Sungho Kang
Yonsei University
Outline

- Introduction
- Anatomy
- FPGA Architecture
- Emulation System
- Case Study
- Conclusion
Challenges for CAD

- **EDA industry**
  - 0.35% of the electronics industry
  - It is a vital enabling technology

- **CAD in service**
  - Not a contest in inventing new algorithms
  - Co-think with designers

- **Technology Trend**
  - Short life time (time-to-market pressure)
  - Submicron processing
  - High degree of system complexity
  - Merging of CPU, DSP, communications, consumer electronics
  - Embedded software: complicated even to specify
  - Reuse of hardware
  - Increased field programmability
Logic Emulator

**Introduction**

- Program through software
- Manufactured in a fab
- REPLACEABLE

**Emulator** → **Chip**
What is Emulation?

Turnkey rapid prototyping systems

- Read users design and automatically partition & map to array of FPGAs
- Enable user to run at system level and verify with application software
- Full internal visibility to debug - thousands of probes
- Modify design in minutes
Bugs Found with Emulation:

- Functional ASIC bugs
- Board/system-level bugs
- Software, firmware bugs
- Synthesis bugs
- Bugs that require rich, real-world stimulus or high throughput to find
- Bugs caused by spec. misinterpretation
Design Flow for Emulation System

Introduction

- Design netlist
- Design Import
- Compile
- System Setup
- Download
- Emulation

Logic Analyzer, Debugger
Comparison with Co-simulation

Performance potential of simulation accelerator is not achievable with current testbench strategies

- Speed of testbench (workstation)
- Channel latency & bandwidth
- Frequency of communication
- Design under test execution speed
Comparison with FPGA

- FPGA
  - High chip capacity
  - Slow compilation
  - Low I/O to gate ratio

- Emulation
  - Fast compile speed
  - Productive debugging
  - High I/O to gate ratio
  - On-board logic analyzer

- Generic FPGAs used for emulation
  - Unpredictable capacity and highly variable routing delays with poor debuggability
Anatomy of an Emulator

- Emulation modules with FPGAs & cross point chip
- Special memory cards for mapping very complex of deep memory
- Instrumentation cards for debug
- Inter-card connection crossbar backplane to allow modular capacity addition
- Specialized add-on cards for cores
- Target interface hardware
- Cable for target system interface or debug
Emulator Architecture

- Hierarchical Multiplexed Architecture Simplifies Design Mapping Process
A logic emulator is a system of:
- Programmable hardware with capacity much greater than one FPGA
- Software which automatically programs the hardware according to a gate level design representation
- Software and hardware to support operation and analysis of the emulated design as a component in real hardware
System Overview : SW Components

- Design compiler
  - Netlist reader and parser:
    - Reads and parses gate-level design netlists
  - Technology mapper:
    - Maps design components into optimal emulator equivalents
  - System-level Partitioner and Placer:
    - Partitions mapped design into boxes, boards, ultimately into FPGA netlists.
  - System-level Interconnect router:
    - Determines the programming of interconnect hardware to complete nets cut by the partitioner
  - FPGA compiler:
    - Reads each FPGA netlist, maps, partitions, places and routes FPGA.
  - Timing Analysis (optional):
    - Analyzes compiled design on emulation hardware for speed, hold violations.

- Runtime download and analysis controller.
- Graphical User Interface Hardware diagnostics
System Overview : HW Components

- **Logic emulation boards**: FPGAs and interconnect chips
- **Memory emulation boards**: RAMs, FPGAs and interconnect.
- **System interconnect board**: chips which interconnect emulation boards.
- **I/O Connectors and Pods**: connects to in-circuit interfaces, external components.
- **Instrumentation**: stimulus generator, logic analyzer, vector interface.
- **Controller**: downloads configurations, operates instruments.
- **Interface**: to host computer.
Why Emulate?

- Concurrent design verification - faster time to market
- Higher predictability of schedule - reduced project risk
- Fewer design changes in final phases - improve quality
- Lower cost for fewer silicon iterations - lower cost
- For mission critical designs - high quality
- Simulation: only methodologies are limited by processing power in verifying complex designs - better verification
- Emulation is the only verification methodology which is keeping up with system complexity
Need for System Level Emulation

- Emulation combines the flexibility of simulation and the realism of a prototype
  - Simulation: limited by the availability of software models
  - Custom prototyping: increased time to build and debug
- Leverages synthesis technology to optimize design differentiation and uniqueness
- Enable fast, incremental design changes that shorten design iteration cycles and improve quality
- Avoid costly respins of silicon and saves months of redesign
- Increased confidence in your entire project schedule and your ability to meet requirements
Design Verification Methodologies

- Simulation: allows observation of a fraction of real world interactions (sequential)

- Emulation: enables the designer to explore alternatives (parallel)
  - Verification takes place in a hardware environment
  - The design is retargeted to a programmable hardware environment

- Rapid prototyping: allows operating speeds such that all interfaces to target applications can operate in real time.
  - If the system runs at real time, the quality of the algorithm can be evaluated on the fly and DSP design time can be greatly reduced.
Advantages of Emulation

- Emulation is the only verification methodology which is keeping up with system complexity

<table>
<thead>
<tr>
<th>Time</th>
<th>Speed up factor</th>
<th>Verification Methodology</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 second</td>
<td>$10^7$</td>
<td>Actual Hardware</td>
</tr>
<tr>
<td>10 seconds</td>
<td>$10^6$</td>
<td>Logic Emulation</td>
</tr>
<tr>
<td>2 minutes</td>
<td>$10^5$</td>
<td></td>
</tr>
<tr>
<td>16 minutes</td>
<td>$10^4$</td>
<td></td>
</tr>
<tr>
<td>3 hours</td>
<td>$10^3$</td>
<td></td>
</tr>
<tr>
<td>1 day</td>
<td>$10^2$</td>
<td></td>
</tr>
<tr>
<td>12 days</td>
<td>$10^1$</td>
<td></td>
</tr>
<tr>
<td>3 months</td>
<td>1</td>
<td>Software Simulations</td>
</tr>
</tbody>
</table>
Advantages of Emulation

- Emulation performance is not a function of design size
Motivation: Verification Cycles

- Simulation is the most compute-intensive problem in EDA

It’s Inside the Design Cycle

Design Capture ➔ Simulate ➔ More Tests!

Correct? ➔ Yes ➔ Design Capture

No ➔ Edit Design ➔ Simulate

Design Sizes are Growing

- 2X Design Size ➔ 2X work / vector.
- 2X Design Size ➔ 2X vectors.

⇒ 2X Design Size ➔ 4X workload
Motivation : Verification Realism

- Often the chip meets its spec, but does not work in the system:
  - Spec errors, misunderstandings:
    - “I thought your chip was handling that...”
  - Real system puts designs into unanticipated situations:
    - Interaction between components across time and function: Combinatorial Explosion
      - i.e. the Ethernet driver interrupts a page fault which is servicing a floating point exception.
  - Other parts of system don’t adhere to their specs, or their specs aren’t known:
    - Undocumented behavior in other devices, such as CPU,
    - Peripherals from other projects or other vendors.
Motivation : Verification Realism (cont.)

• Some applications need real-time operation for verification:
  ▪ Display are far easier to verify by actual observation
  ▪ Closed-loop operation with analog hardware
  ▪ Electro-mechanical controllers
  ▪ Human perception: audio, video compression, processing

• Simulation generally requires test vector development:
  ▪ Costly and difficult, critical path in schedule,
  ▪ Verification depends of test vector correctness,
  ▪ Test vectors may have to be based on assumptions,
  ▪ Test vectors are intrinsically open-loop.
Motivation : Verification Realism (cont.)

- Only when the real design is running its real application in its real environment is correctness assured.

- Emulated design connected to actual hardware can run:
  - actual diagnostic code, compatibility tests,
  - actual operating systems,
  - actual applications,
  - receiving real data from storage, sensors, devices,
  - sending real data to storage, devices, displays.
Motivation: Visibility

- Once a chip is fabricated, placed in a system, and fails
  - Internal probing is impossible
  - It may be difficult or impossible to put the simulation into the failing state for analysis
- Emulated design can have internal probes programmed in, for direct connection to instrumentation
- Emulated design may be used to generate test vectors for fabrication
Rapid Prototyping

• Once emulated design is debugged, it is available for immediate use by software developers.
  ▪ This can directly reduce the project’s critical path and time-to-market.

• Emulated design is available for demonstration to customers, users, management.
  ▪ Proof of concept, proof of progress.
  ▪ Find out early whether result will be satisfactory.

• Architectural workbench:
  ▪ Drive emulation with RTL-level synthesis.
  ▪ Experiment with architectural features on real code and data:
    ◦ structures, sizes, algorithms of caches, busses, buffers,
    ◦ quantity and design of functional units,
    ◦ novel architectures, representations, algorithms, etc., etc.,
Disadvantage of Logic Emulation

- Hardware emulation system is required
- Speed is 5-10 X slower than real design speed
  - System emulation speeds of 1 to 4 MHz are common today
  - Target system must be slowed down for emulation
- Delays do not match those of real design
  - Timing-induced errors are possible, that is, hold-time violation
  - Delay independent functionality may not operate correctly
Logic Emulation

• FPGA-based Hardware Emulation
  ▪ Contain a large pool of general purpose logic block
  ▪ Design preparation time and compilation time are costly

• Processor-based Hardware Emulation
  ▪ An array of basic CPUs or simple Boolean processors that perform basic logic operation on a time sharing basis
  ▪ Design under verification is converted into a simulation data structure, similar to that of a software simulator
  ▪ Slower than FPGA-based
FPGA Architectures for Emulators

- Partitioning, placement and routing optimized for the emulator performance
- FPGA efficient interconnection technology
  (currently use ~25% of FPGA logic gates)
- Interconnection of logic blocks, of multiple FPGAs, of multiple emulator modules
- Incremental design change
- Observability and controllability of debug process
- Memory resource
  (separate memory or FPFA RAM)
- Clock lines
  (low skew, no setup and hold time violations)
- Lower cost (than silicon)
Interconnect Problem

- It is critical to maximize gate capacity and speed by packing as much logic into each FPGA as possible.

- Interconnect hardware architecture must:
  - provide successful connectivity in all cases.
  - Permit maximum logic utilization of the FPGAs,
  - with minimum added delay and skew,
  - at minimum hardware cost.

- Rent’s Rule applies:
  - Observation of Rent at IBM in 1960’s:
    - Pincount of arbitrary subpart of a digital system is proportional to a fractional power of the gatecount.
    \[ P = K \times G^r \]
  - Example in high-performance systems:
    - pins = 2.5 * gates ^ 0.56
Interconnect Problem

- Commercial FPGAs are sized for engineered applications:
  - Designed is designing for FPGA structures.
  - Designer architects system so that subparts fit into FPGA pincounts.
  - Vendors design FPGAs accordingly.

- Emulated designs are completely arbitrary:
  - Structures are not optimum for FPGAs
  - FPGA size and pincount is arbitrary.
  - FPGA subparts are automatically extracted by partitioner.
  - Result:
    - FPGAs are pin-limited, not gate-limited.
    - Logic emulator gets 20-30% as much gate utilization as ordinary FPGA applications.
  - There is a challenge for interconnect architecture and software to maximize gate utilization FPGA pincount.
Field Programmable Interconnects

- **Aptix FPIC**
  - A place and route architecture (not a crossbar)
  - Routing delay not controllable
  - 940 user programmable I/Os

- **IQ 160**
  - 176X176 crossbar
  - Every port can be configured to connect to any port
  - Routing delay is predictable

- **FPGAs**
  - A place and route architecture
  - Routing delay not entirely controllable
  - Typically < 300 programmable I/Os
Virtual Wires (IBM)

- Increase bandwidth by multiplexing
  - > 80% gate utility, but decrease emulation speed.
Partial Crossbar Interconnect

- Useful with the ability of FPGAs to freely assign pins
- Each small full crossbar chip is connected to the same subset of pins on each logic chip
- To configure the system, logic partitioning, placement, routing are performed
- Placement is insignificant, since the interconnection is symmetrical
- Interconnect router is a simple repetitive table-driven task
MARS (Quickturn)

- MARS System from PIE Design
  - Based on Xilinx XC4005 and XC4003
  - Partial crossbar system
  - 500K gates in one box
  - Emulated 2 million gates design
Realizer (Quickturn)

- Automatically configures a network of FPGAs to implement large digital logic designs
- Logic and interconnections are separated to achieve optimum FPGA utilization
- Interconnection by partial crossbars reduces system-level complexity
  - Achieves bounded interconnection delay
  - Scales linearly with pin count
  - Allows hierarchical expansion in a fast and uniform way
- Applications
  - Prototyping for real time verification and operation
  - High-speed simulation accelerator
  - VHDL-driven architecture workbench
Enterprise Emulation System

330K gates + 64 MB memory Enterprise system

- Precision Emulation Software™
- Electronically Programmable backplane bus
- Debug environment & software
- Built-in Logic Analyzer & Stimulus Generator
- Memory debug software with Memory Emulation Modules
- Built-in Workstation interface for co-simulation
Enterprise Emulation System

- Expandable system supports up to 11 independent Logic Emulation Modules (30,000 to 330,000 logic gates)
- System support up to 32 Memory Emulation Modules (Total 64 Mbytes, including multiport RAMs)
- Precision Emulation Software accurately synthesizes even the most complex design
- EDA system integration tools smooth the transition to emulation
- Fast, incremental design change capability shortens design iteration cycles
Enterprise Emulation System

Gate-level Netlist Files

Libraries

Netlist Reader and Parser

Technology Mapper

System-Level Partitioner and Placer

System-Level Interconnect Router

FPGA Compiler

FPGA Compiler

FPGA Compiler

Interconnect Compiler

Configuration Files for Each Logic and Interconnect Chip

Design Database

Timing analysis (Optional)
Rapid prototyping is made possible by
- FPIC (Field Programmable Inter Connect)
- FPCB (Field Programmable PCB)
- FPGA
- Synthesis

RAPID PROTOTYPE

- High Speed System Operation
- Standard Component Integration
- Re-configurable, Automation
- Architectural Target Definition

- Design Style Enforcement
- Large Logic Handling
- software Automation
- Library Rotargeting

- Large Logic Capacity (20K)
- High Speed Feasible (30MHz)
- Re-configurable, Automation

Programmable Hardware
COBRA Prototyping Environment

- Connector: 90 pins/side, 75 bit link to two neighbors
- 4 base modules (one with RAM module)
- Supporting software
  - Partitioning, synthesis, hardware debugging software

![Diagram](image)

CTRL
Xilinx 4025
Xilinx 4025
Xilinx 4025
Xilinx 4025
CTRL
ROM
ROM
Splash 2

- Based on Xilinx XC4010
- Scalable from 16 to 256 processing elements
  - (1 ~ 16 boards)
- Higher I/O bandwidth (DMA from Sun SBus)
- Increased connectivity
- (linear data path + crossbar)
- Application programs in behavioral VHDL
- Array board:
  - 16 processing elements
  - 16 * 16 crossbar switch
  - 0.5 Mbyte memory
  - 36 bit bi-directional data paths
Case Study

Computer System Integration Stages

Equivalent Real-Time System Cycles

- 100000 sec
- 10000 sec
- 1000 sec
- 100 sec
- 10 sec
- 1 sec
- .1 sec

Current Simulation Capability

- Run Applications
- System Level Diagnostics
- Boot Operating System
- BIOS Self Test
- Emulation Domain

Computer System Integration Stages

- Chip
- Chip Set
- System Hardware
- System Hardware & Software

Equivalent Real-Time System Cycles
UltraSPARC Emulation

Case Study

- Low-skew clock distribution
  - The vendor supplied clocktree analysis procedure
  - Gated clock are redesigned to either remove the gating or pull the gating to the root to allow use of low-skew nets

- Possible problem areas
  - Gated clocks (e.g. scan control)
  - Feed-thrus (designed to reflect the silicon floorplanning)
    - Only actual direct connections are important
  - Repeaters: Emulation does not need repeaters
  - Precharge logic & pass-thru latches must be redesigned into a static representation (careful verification required)
UltraSPARC Emulation

Case Study

- Verification at the block/megacell level
  - All the reimplementation of megacells (due to internal clocking or memory) must be carefully verified for cycle accurate functionality

- Full chip configuration
  - takes 36 walk clock hours using 5 fast +70 computers
  - 5 emulation systems form Quickturn
  - 50 system to system cables

- Testbed
  - Over 5000 nets routinely probed (128k depth)
  - The internal & external logic analyzer samples are Integrated

- Return on emulation investment
  - Pre-tapeout : 25% (additional final design verification)
  - Post_tapeout to pre-silicon : 25% (stress testing)
  - Post-silicon : 50% (visibility for debugging, what-if experiments)
Motorola 68060

• Methodology
  - Generating a gate level model
    - Handling custom logic and memory arrays
    - Vector generation and verification
  - Emulation library development
  - Emulation model build
  - Functional emulation
  - In-circuit emulation

• Time comparison (10^8 vectors)
  - Compiled RTL simulator : 6weeks
  - Functional emulation (1KHz) : 25hrs
  - In-circuit emulation (1MHz) : 90secs
  - 68060 silicon (50MHz) : 1sec
PowerPC

Case Study

- Demands:
  - PowerPC 603/604™ bus speed:
    - 2~4 times the speed of the external bus.
  - Pipeline controller to run at 75MHz

- CPLD is a better choice than FPGA
  - CPLD provides a higher number of pterms in each cell
  - The state bits are visible
  - CPLDs boast a very predictable delay path
Design for Emulation

- Synchronous design
- Pipeline design techniques
- Short arithmetic functions (minimize logic level)
- Minimize bit width
- Use of I/O FFs where possible (latches)
- Careful mapping between functions and FPGAs
- Minimize high fanout net or use available buses
- Use of small blocks (20-40K gates)
- Care when gating the clock tree (e.g. low power)
- Limit module I/O count
Synthesis to Support Emulation

- Synthesis to compile behavioral models for emulation

---

**Synthesis for Emulation**

1. **Specification (Detailed RTL)**
   - Logic Synthesis
     - NetList
       - Placement Routing
         - Layout
           - Timing Analysis
2. **Synthesis for Emulation**
   - Formal Verification
   - Model Debugging
   - Functional Abstraction
3. **Emulation**
   - Emulator

---

**Model**

- LDS (proprietary HDL) subset of VHDL
- CLOSE Synthesis for Emulation
  - CLB-netlist
  - META Systems processing chain

---

**Case Study**
Application of Synthesis

- Efficient models to handle specific logic styles
  - Multiple clocks
  - Multiple input latches
  - Tristate and precharged signals
- Efficient logic optimization
- Module generators for specific operators
- Control of visibility and controllability
Conclusion: Ideal Emulation

- **Cheaper**: $1.25/gate in 1994
- **Easier to use (reduce time-to-emulation)**
  - Automatic design transformation (synchronous design: setup/hold time, gated clock, wired logic, precharge logic)
  - Compiler (partitioning and mapping)
  - Hierarchical modular architecture
  - Target interface for complex multi-chip system
- **Powerful debug environment**
  - Debugging environment or short debug turn-around time
  - Modular complication for incremental change
  - Powerful logic analysis
- **Scalability to handle increasing complexity**
  - Modular and flexible packaging
- **Current Technology**
  - 20-30M emulation gates, over 5MHz emulation speed