

EE290 A: Advanced Topics in CAD
Component Based Design
of Electronic Systems

Professors Kurt Keutzer and Richard Newton
Department of Electrical Engineering and Computer Sciences
University of California at Berkeley
Spring 1999

Outline

• *Raw trends*
• Implication of trends - SOC/component based design
• Challenges in component based design
• Homework 1
**Semiconductor Capital Investment**

- **Year of Investment**: 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 0
- **Investment ($B)**: 0, 10, 20, 30, 40, 50, 60, 70

- **Asia**
- **Europe**
- **Japan**
- **USA**

Source: Dataquest

---

**Moore’s Law**

- **Transistors**: 4004, 8086, 8080, 68000, 68020, 80386, 80486, 68040, Pentium, Pentium Pro, PPC601, PPC603, MIPS R4000

- **Microprocessors**: 10x/6 years

NTRS: Microprocessor: total transistors/chip

NRTS: ASIC/Microprocessor logic transistors
**NRTS: Chip Frequency (Ghz)**

![NRTS Chart](image)

- On-chip, global clock, high performance
- On-chip, local clock, high-performance

---

**Evolution of the EDA Industry**

<table>
<thead>
<tr>
<th>Results</th>
<th>What’s next?</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Design Productivity)</td>
<td></td>
</tr>
<tr>
<td>1978</td>
<td>1985</td>
</tr>
<tr>
<td>1985</td>
<td>1992</td>
</tr>
<tr>
<td>1992</td>
<td>1999</td>
</tr>
<tr>
<td>Synthesis - Cadence, Synopsys</td>
<td></td>
</tr>
<tr>
<td>Schematic Entry - Daisy, Mentor, Valid</td>
<td></td>
</tr>
<tr>
<td>Transistor entry - Calma, Computervision</td>
<td></td>
</tr>
</tbody>
</table>

**McKinsey S-Curve**

<table>
<thead>
<tr>
<th>Effort</th>
</tr>
</thead>
<tbody>
<tr>
<td>(EDA tools effort)</td>
</tr>
</tbody>
</table>
To Design, Implement, Verify 10M transistors

Staff Months

- 62.5
- 125
- 625
- 6250
- 62,500

Implementations here are often not good enough
Because implementations here are inferior/unpredictable

Why?: The Deep Sub-Micron Double-Whammy

Today’s Design Methodologies Will Not Scale Much Further

- The Deep Sub-Micron (DSM) Effect ($\leq 0.25\mu$)

$\propto$ DSM

“Microscopic Problems”
- Wiring Load Management
- Noise, Crosstalk
- Reliability, Manufacturability
- Complexity: LRC, ERC
- Accurate Power Prediction
- Accurate Delay Prediction
- etc.

$\propto 1/\text{DSM}$

“Macroscopic Issues”
- Time-to-Market
- Millions of Gates
- High-Level Abstractions
- Reuse & IP: Portability
- Predictability
- etc.

Everything Looks a Little Different

...and There’s a Lot of Them!

Adapted from Dr. Kurt Keutzer, Synopsys
Stolen back by Prof. Kurt Keutzer, UC Berkeley
**Crisis: Productivity Gap**

![Graph showing productivity gap over time with logic transistors per chip and productivity growth rates.](graph.png)

**Within a 50K - 100K Module**

- 75 - 100% delay in gates
- 3.5µ - 1.0µ (1980 - 1990)
- 0.18µ - 0.1µ (1998 - 2005)

This flow should work OK for blocks of 50-100K gates and should continue to work OK in the future.

Proper sizing within flow is absolutely required:
- typically: size down then up
- try: size up- then down
Outline

- Raw trends
- Implication of trends - SOC/component based design
- Challenges in component based design
- Homework 1

Design Paradigm: Re-useable IP

Achieve required design productivity by assembling re-useable blocks of intellectual property (IP)

Silicon capability enables integration of entire systems on a single die
Also: Trend towards programmable Solutions

Key Forces Driving Component-based Design

- Exponentially increasing complexity device complexity/capability
- Productivity / Time-to-market requirements more stringent
- Synthesis from very high-level descriptions does not provide adequate quality-of-results
- Trends toward programmable solutions
Next Step in the Evolution of the EDA Industry

1978 Transistor entry - Calma, Computervision
1985 Schematic Entry - Daisy, Mentor, Valid
1992 Synthesis - Cadence, Synopsys
1999 Component-based design

McKinsey S-Curve

Growing list of IP Blocks

Video: MPEG, DVD, HDTV
Audio: MP3, voice recognition
Processors: CPUs, DSPs, Java
Networking: ATM, Ethernet, ISDN, FibreChannel, SONET
Bus: PCI, USB, IEEE 1394
Memory: SRAM, ROM, CAM
Wireless: CDMA, TDMA
Communication: modems, transceivers
Coding: speech, Viterbi, Reed-Solomon
Display drivers/controllers: TFT
Other: sensors, encryption/decryption, GPS

Power PC core: 3.1 mm² in 0.35µ
ARM Core: 3.8 mm² in 0.35µ
MPEG2 Decoder: ~65k gates
PCI Bus: ~8k gates
Ethernet MAC: ~7k gates (soft)
RSA Encryption: ~7k gates
Example: Configurable Microprocessor

Describe the processor attributes from a browser-like interface

Using the tensilica processor generator, create...

Customized Compiler, Assembler, Linker, Debugger, Simulator

Use a standard cell library to target to the silicon process

Easily select Instruction Set attributes

Tensilica Processor Generator

Instruction Set Architecture
- MAC with 8-bit accumulator
- Exception options
- High priority interrupt options
- Timer interrupt options
- Floating point option

Exception
- User Exception Vector
- Kernel Exception Vector
- Illegal Instruction Handler Address
- System Call Instruction Handler Address
- Load-Store Error Handler Address
- Register Window Overflow Handler Address
- Register Window Underflow Handler Address
- Reset Vector
- IP Exception Handler address
Set Memory and Cache attributes

Outline

- Raw trends
- Implication of trends - SOC/component based design
- Challenges in component based design
- Homework 1
**Challenges in Component-based Design**

*What is a component - what is the right quanta/granularity of capability?*

**Design**
- How are components described?
- How are they modeled?

**Implementation**
- How do we trade-off between HW and SW implementations?
- In HW how do we trade off between soft, firm, and hard macros?

**Verification**
- How do we verify individual components?
- How do we verify component interfaces?
- How do we verify a family of parameterizable instances?

---

**What is a Component?**

- A *component* is defined as any fully encapsulated behavior.
  - It is analogous to an object in object-oriented programming in that it may have an associated state, behavior, and identity.
  - We use the term component, rather than object, to stress the fact that the implementation medium—logic, memory, software, reconfigurable logic, or some combination—is not a factor in the specification of the component itself.
What is a Component?

- Access to components is provided via an interface and the interface is the only way to interact with the component.

What is a Component?

- A system is defined as one or more, possibly interacting, components and their associated environment.
- The environment specifies all of the external constraints and all possible inputs the collection of components might be asked to respond to and so closes the system.
Challenges in Component-based Design

What is a component - what is the right quanta/granularity of capability?

Design
- How are components described?
- How are they modeled?

Implementation
- How do we trade-off between HW and SW implementations?
- In HW how do we trade off between soft, firm, and hard macros?

Verification
- How do we verify individual components?
- How do we verify component interfaces?
- How do we verify a family of parameterizable instances?

The Seven Views of Computer Systems

View One: Structural Levels of a Computer System
View Two: Levy’s Levels of Interpreters
View Three: Packaging Levels of Integration
View Four: A Marketplace View of Computer Classes
View Five: An Applications/Functional View of Computer Classes
View Six: The Practice of Design
View Seven: The BLAAUW Characterization of Computer Design
**View One: Hierarchy of Computer Levels**

Adapted from Bell and Newell [1971]

**View Two: A Hierarchy of Interpreters**

[Levy, 1974]
Design: How are Components Described and How are They Modeled?

Today, most “software” components are described using either assembly language or a C model

- Compiled and executed using C development environment for a target processor ISA
- Semantics defined operationally by the compiler/assembler and “language extensions” via packages and system calls

Design: How are Components Described and How are They Modeled?

Today, most “hardware” components are described using a “C model”

- Compiled and executed using C development environment
- Usually an “untimed” model
- Central issues: handling concurrency, special data types, language subsets, language extensions via packages

In certain application-specific areas, other approaches are more common (e.g. SPW, Cosap, Matlab)

- Usually embodies a particular model of sequence/time and specifies a particular path to implementation
- New “general-purpose” approaches under development and research
- CoWare, Felix, Polis, Ptolemy-2, JavaTime, …

Central Issue: Relationship of Description/Specification to final implementation
**Design: How are Components Described and How are They Modeled?**

At lower levels of abstraction:
- **Register Transfer Level (RTL):** VHDL, Verilog
- **Gate Level:** Vendor gate library (NAND, flip-flop, etc.)-schematic
- **Physical:** Mask layout (rectangles on layers)

Ways of delivering SOC IP:
- **Hard:** Detailed and fully-characterized layout in a specific process
- **Soft:** RTL Level in Verilog or VHDL; “implementation independent”
- **Firm:** Soft + a collection of constraints and requirements for the implementation

---

**What is an Architecture?**

- Amdahl, Blaauw, and Brooks, 1964, defined three interfaces:
  - **Computer Architecture:** “The attributes of a computer as seen by a machine language programmer.”
  - **Implementation:** “Actual hardware structure, logic design, and datapath organization.”
  - **Realization:** “Encompasses the logic technologies, packaging, and interconnection.”

- Hennessy and Patterson
  - **Instruction-Set Architecture:** programmer-visible instruction set. Serves as a boundary between the hardware and the software.
  - **Organization:** High-level aspects of a computer’s design, including the memory system, bus architecture, and internal CPU design
  - **Hardware:** Used to refer to the specifics of a machine. Including detailed logic design and packaging technology
  - **Architecture:** covers all three aspects.
What is an Architecture?

- A description of the behavior of a system that is independent of its implementation.
  - Isomorphic to its "interface specification" (Siewiorek, Bell, Newell, 1971)
  
  For example:
  - Instruction set definition of a computer
  - Z-domain description of a filter
  - Handshaking protocol for a bus

- May guide implementation (contain 'hints' or 'pragmas')
  - e.g. a particular specification may lead naturally to a serial or parallel implementation.

What is an Instruction-Set Architecture?

- Example: Instruction set

  | Inst_A     | Inst_A     |
  | Inst_B     | Inst_B     |
  | Inst_C     | Inst_C     |
  | ...        | ...        |
  | Inst_C may not follow Inst_A | Inst_C may not follow Inst_A because Inst_A uses a scratchpad register that Inst_C will over-write. |

Architecture

Not architecture
**Specification vs. Description**

**Specification:** Saying what I want; describes behavior in terms of results.

\[ \forall A \{ A[i,j] \leftarrow 0 \} \]

**Description:** Saying how to do it; describes behavior in terms of procedure or process.

\[
\begin{align*}
\text{for}(i=0; & i<N; i++) \\
\text{for}(j=0; & j<M; j++) \\
A[i][j] & = 0;
\end{align*}
\]

We do not have specification languages for general-purpose digital design. For some special-purpose applications (e.g. DSP) we do.

---

**Languages versus Models**

- **Estere**
- **Language**
- **UML**
- **Synchronous**
- **Reactive**
- **FPGA-Based**
- **Implementation Media**
- **Model of Computation**
- **C for RTOS**
- **RTOS**
Languages versus Models

- Esterel
- “C, C++”
- VHDL
- Matlab
- UML

Synchronous
Reactive

Discrete
Event

“FSMs”

Hard-Wired
ASIC

C for
RTOS

FPGA-Based

“Von Neumann”
Processor

SLSMSLM

MLSM

(really just syntax)

FPGA-Based

“Von Neumann”
Processor

Hard-Wired
ASIC

RTOS
VHDL: The “nroff/latex” of Design

In Most HDLs, “wires” are declared but the passage of time is embedded in the control structures.

We are caught up (once again!) with imperative, sequential thinking and a Von Neumann model.

We need a way of capturing both temporal and spatial encoding in a single, unified mathematical model.

Use a type mechanism: “t-types”
**Challenges in Component-based Design**

What is a component - what is the right quanta/granularity of capability?

Design
- How are components described?
- How are they modeled?

Implementation
- How do we trade-off between HW and SW implementations?
- In HW how do we trade off between soft, firm, and hard macros?

Verification
- How do we verify individual components?
- How do we verify component interfaces?
- How do we verify a family of parameterizable instances?

---

**Architecture for SOC**

- Application API
- SOC Block Integration Layer
  - Coarse-Grain API (thread level)
  - General-Purpose Processor with BIOS and OS
  - Vector Processor Interface
  - Multi-User Interface
A Vision: Design System 2010

- Specification
  - Constraints: Real time, PDA, Precision Communication Complexity
  - Threader
    - (1) Threads mapped to processing units
  - Thread Optimization & Implementation
    - Processing Unit
      - Microarchitectural Optimization
        - COTS Silicon
          - FU
          - CU
          - Memory
          - MIL-IP
          - Logic
          - FU
          - FU


- Embedded µP Core
- 90K CBA gates
Transition to Extensive Use of Regular Structures

Move Towards Regularity and Programmability

Regular logic
Programmable structure
Other (random logic, etc.)
Particular Function (e.g. MPEG)

- ASIC
- GP processor
- X-GP processor

Time

Real-time requirement

Non real-time

Example of System Behavior

From Alberto Sangiovanni-Vincentelli
IP-Based Design of Behavior

Transport Decode

Rate Buffer

Sensor Synch

Control

Frame Buffer

Video Output

User/Sys Control

Mem

User Interface

Written in C

From Alberto Sangiovanni-Vincentelli

IP-Based Design of Implementation

Which Bus? P1? AMBA? Dedicated Bus for DSP?

Can I Buy an MPEG2 Processor? Which One?

Do I need a dedicated Audio Decoder? Can decode be done on Micro-controller?

Which DSP Processor? C50? Can DSP be done on Micro-controller?

Which Micro-controller? ARM? HC11?

How fast will my User Interface Software run? How Much can I fit on my Micro-controller?

From Alberto Sangiovanni-Vincentelli
Separate Behavior from Architecture

System Behavior
- Functional Specification of System.
- No notion of hardware or software!

Implementation Architecture
- Hardware and Software
- Optimized Computer

Map Between Behavior and Architecture

Audio Decode Behavior Implemented on Dedicated Hardware

Transport Decode Implemented as Software Task Running on Micro-controller

Communication Over Bus
**Basic Principles**

Design Methodology general enough to capture most of application domains

Existing methods should be modeled to allow transition

Powerful policies to allow fast development of complex systems with correctness guarantees

Policies supported by verification and synthesis tools both at the abstract and at the physical level

Constraint-based Approach to generate appropriate guiding principles for Subsequent Implementation steps

Validation of ideas can only come by applying it to “real” designs

From Alberto Sangiovanni-Vincentelli

---

**System Level Design**

Design Methodology:

- **Top Down Aspect:**
  - *Formalization:* precise unambiguous semantics
  - *Abstraction:* capture the desired system details
  - *Decomposition:* partitioning the system behavior into simpler behaviors
  - *Successive Refinements:* refine the abstraction level down to the implementation by filling in details and passing constraints

- **Bottom Up Aspect:**
  - IP Re-use (even at the algorithmic and functional level)
  - Components of architecture from pre-existing library

From Alberto Sangiovanni-Vincentelli
System Design

Estimation and Modeling

Combines Behavioral Parameters and Architectural Models
Communication Refinement

Separate Function of blocks from inter-block Communication

Substitute lower-level detail for communications behavior

From Alberto Sangiovanni-Vincentelli
**Challenges in Component-based Design**

What is a component - what is the right quanta/granularity of capability?

**Design**
- How are components described?
- How are they modeled?

**Implementation**
- How do we trade-off between HW and SW implementations?
- In HW how do we trade off between soft, firm, and hard macros?

**Verification**
- How do we verify individual components?
- How do we verify component interfaces?
- How do we verify a family of parameterizable instances?
**System-on-a-Chip Integration Flow**

- Models
  - Bus Functional Models
  - Timing Models
  - Power Models

**System Abstraction**
- System Physical Floorplan
- System Hardware/Software Partitioning
- System Design Planning

**System Verification**
- Functional
  - Test
  - Noise
  - Reliability
- Timing
- Power
- Physical
- Manufacturing
- Software

---

**SoC Model Views**

- Core Creation
  - Core Design
    - Automatic Model Creation
    - Post-Design Remodeling

- Core Views
  - Testbench/Tests
    - Instruction Set Architecture
    - Bus Functional
    - Full Function Cycle
    - Timing Model
    - Full-Function with Timing
    - Test Model
    - Floorplan/Phy Model
    - Electrical Rule Model

- Integration
  - Functional Design
  - Testbench Design
  - Core Creation
  - Core Views
  - Integration
  - Place & Route and Chip Finishing
Berkeley Wireless Research Center (BWRC)

Conventional cellular phone solution

- Research into technology and design methodologies for CMOS single chip radios
- Exploring future applications of wireless technology, 4th generation and beyond

BWRC Long-Term Research Drivers

Universal Radios for 4th Generation
- Two generations beyond present digital cellular
- Low energy, high-performance programmable computing platform
- Systems and circuits focus to resolve rules of engagement at the air interface and to allow for peaceful co-existence

Picoradios
- Ultra-low power, low cost, embedded CMOS radio’s ( < 1 mW)
- EDA systems for rapid, optimized implementation

Ultra-High Bandwidth Millimeter Radios
- Scaled CMOS solutions for 20 - 60 GHz operation
- Architectures and Device modeling
Homework 1: JPEG as a Component

You are to evaluate/estimate one of a number of possible implementations of the JPEG specification provided on the course web page.

You will estimate, as accurately as you can and with as much justification as you can:

- Frames/second
- Frames/mW
- Production cost of your solution

You may work in groups of one or two

Your results should be submitted online by Friday, 1/29, 5pm

We will compare results and assumptions in Week 3

---

Scenario 2 Implications: Power as the Driver

We believe power always has been the driver!

Source: R. Brodersen, Berkeley
**Challenges in Component-based Design**

What is a component - what is the right quanta/granularity of capability?

**Design**
- How are components described?
- How are they modeled?

**Implementation**
- How do we trade-off between HW and SW implementations?
- In HW how do we trade off between soft, firm, and hard macros?

**Verification**
- How do we verify individual components?
- How do we verify component interfaces?
- How do we verify a family of parameterizable instances?