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

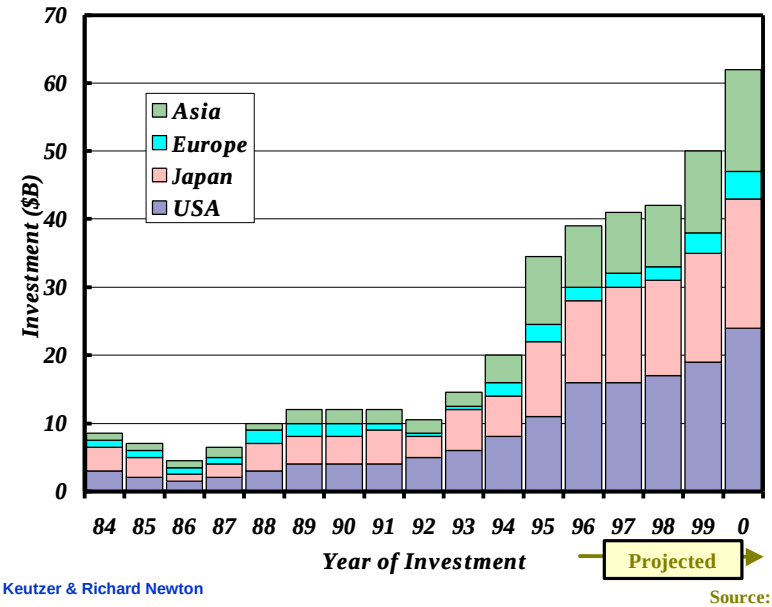
University of California at Berkeley

Spring 1999

Outline

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

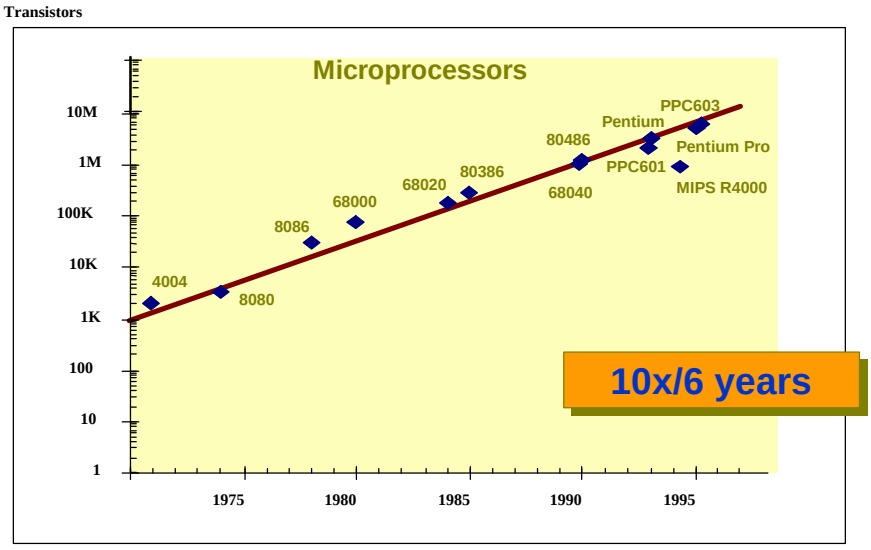
Semiconductor Capital Investment



Kurt Keutzer & Richard Newton

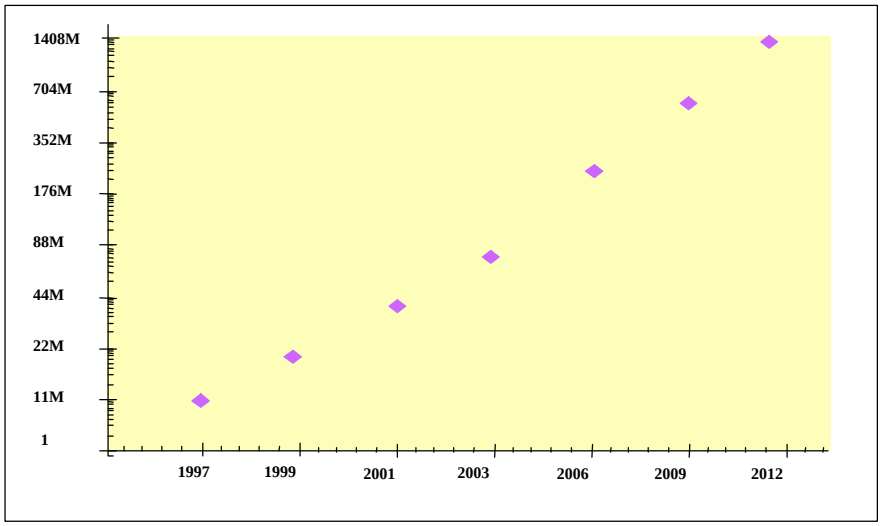
Source: Dataquest

Moore's Law

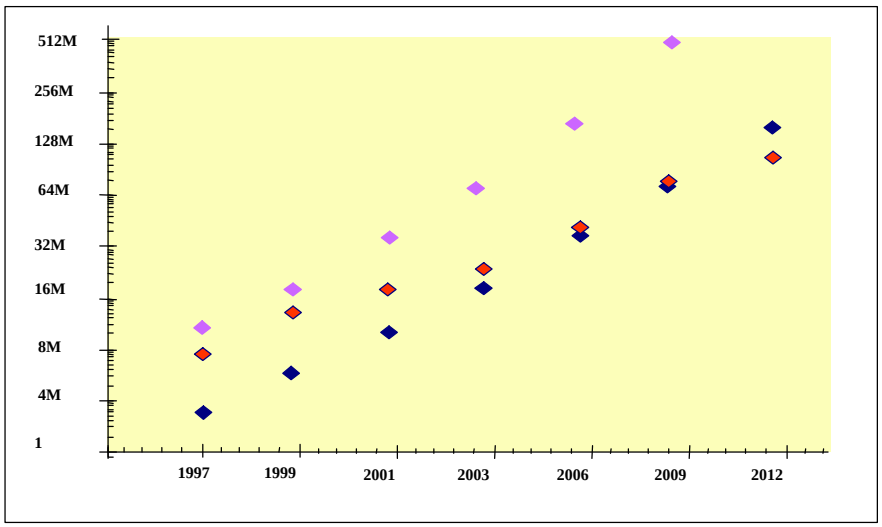


Kurt Keutzer & Richard Newton

NTRS: Microprocessor: total transistors/chip

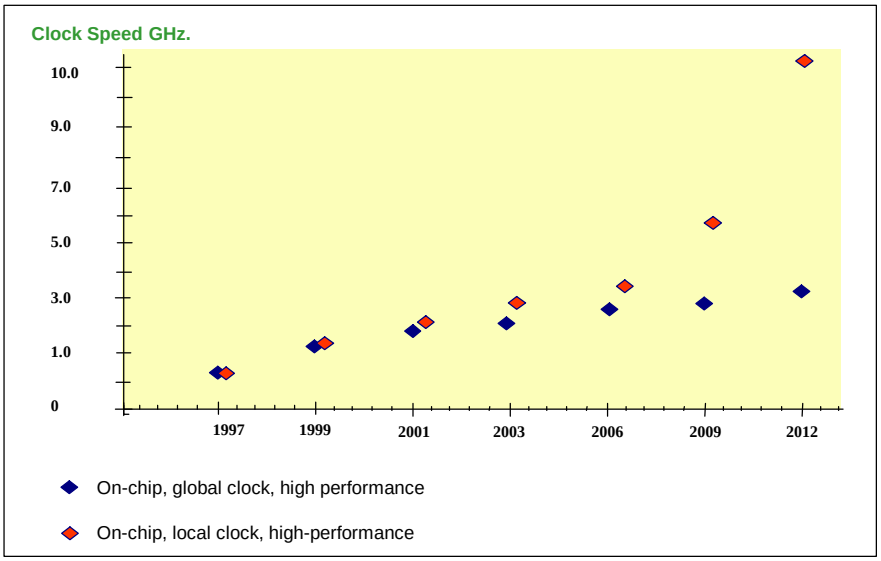


NRTS: ASIC/Microprocessor logic transistors

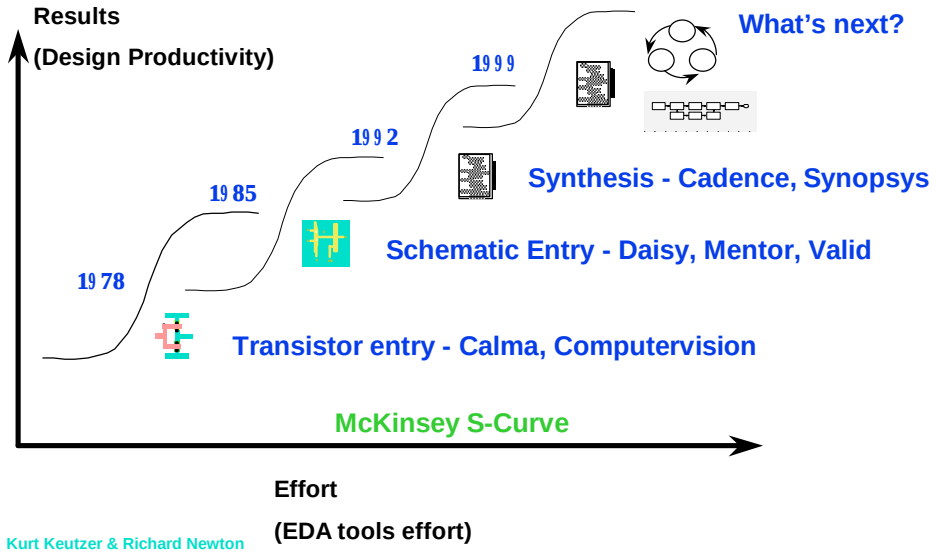


Total microprocessor tr. ◆ Microprocessor logic tr.cm2 ◆ ASIC logic tr. cm2 ◆

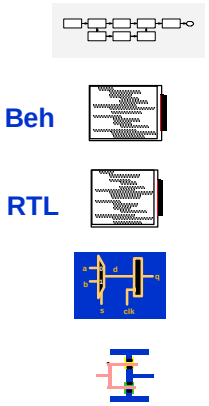
NRTS: Chip Frequency (Ghz)



Evolution of the EDA Industry



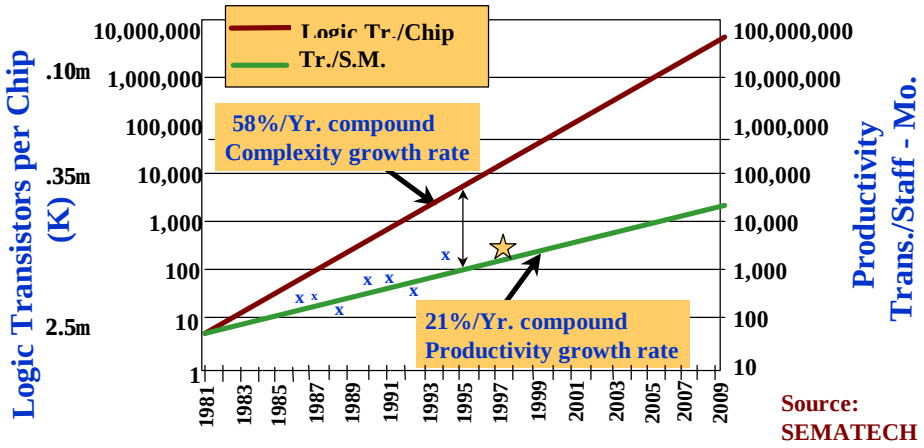
To Design, Implement, Verify 10M transistors



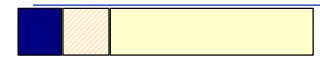
Why?:The Deep Sub-Micron Double-Whammy

Stolen back by Prof. Kurt Keutzer, UC Berkeley

Crisis: Productivity Gap

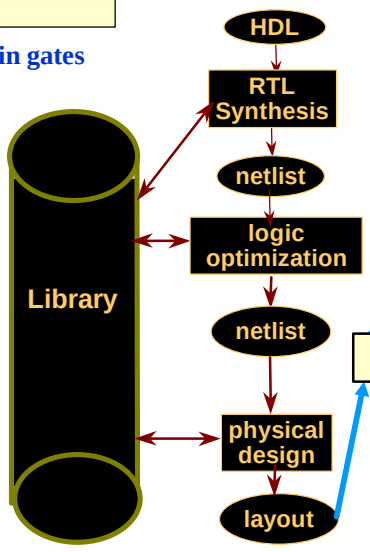


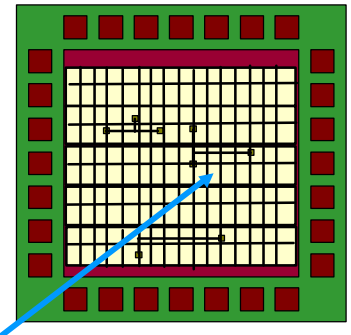
Within a 50K - 100K Module



75 - 100% delay in gates
3.5µ - 1.0µ
1980 - 1990
.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

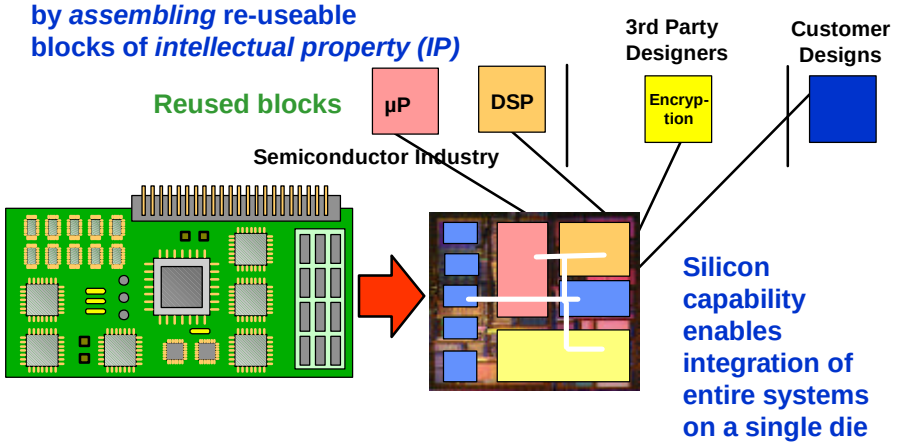
- typically: size down then up
- try: size up- then down

Outline

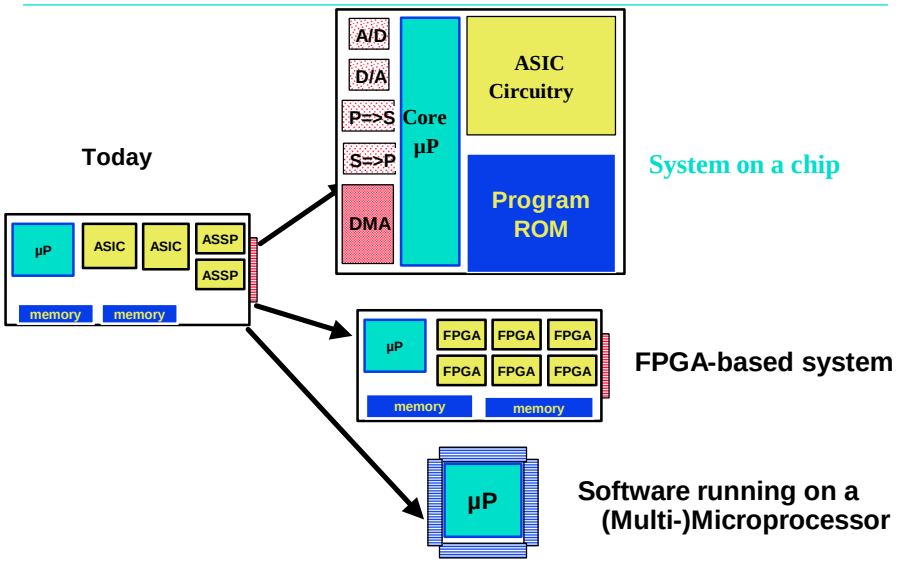
- **Raw trends**
- ***Implication of trends - SOC/component based design***
- **Challenges in component based design**
- **Homework 1**

Design Paradigm: Re-useable IP

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



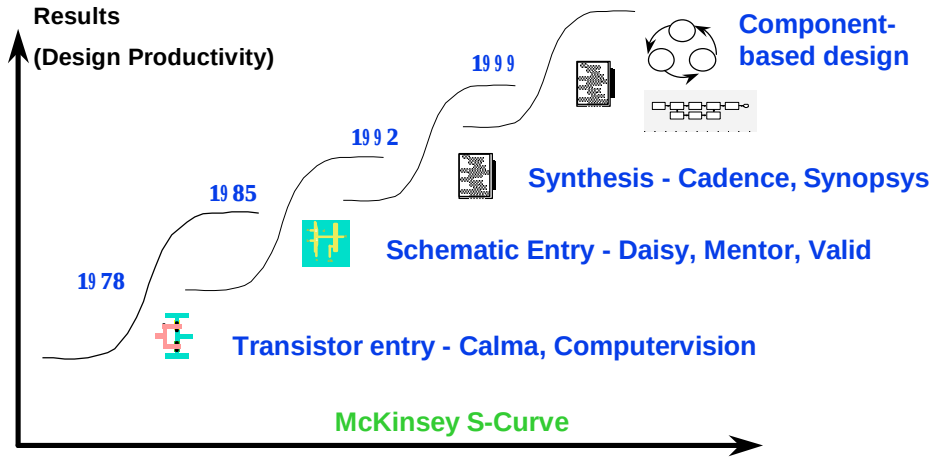
Also: Trend towards programmable Solutions



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



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: 3.1mm^2 in 0.35µ

ARM Core: 3.8 mm^2 in 0.35µ

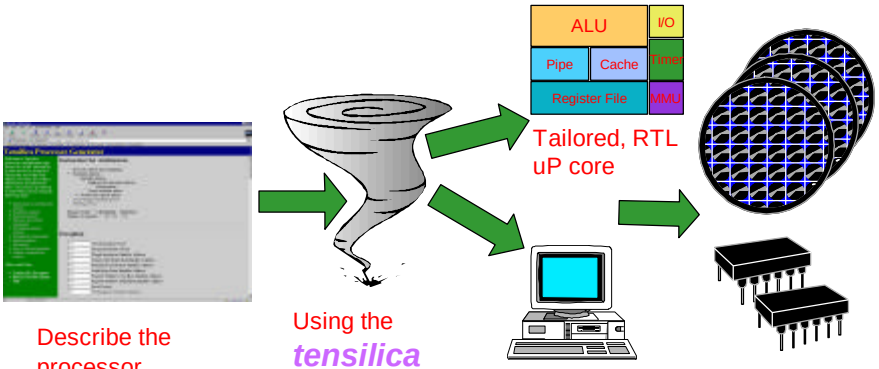
MPEG2 Decoder: ~65k gates

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

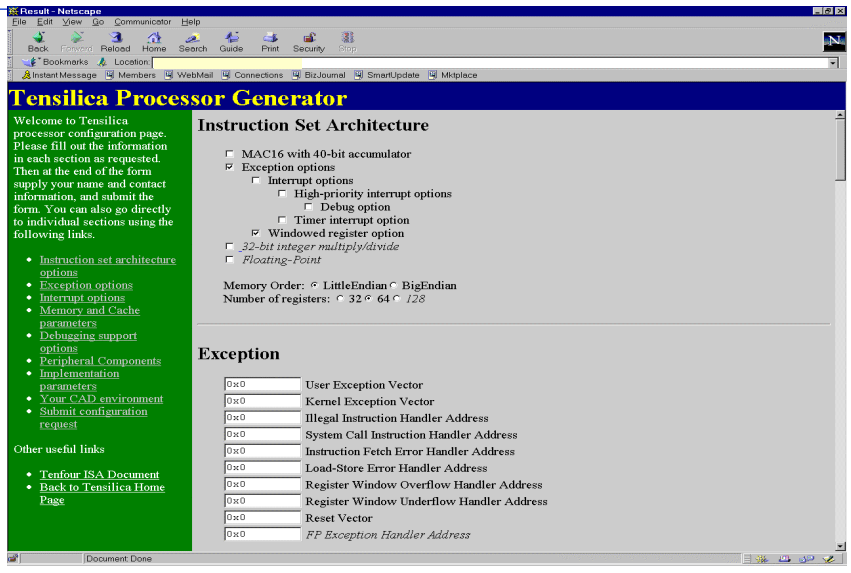
Using the *tensilica* processor generator, create...

Tailored, RTL uP core

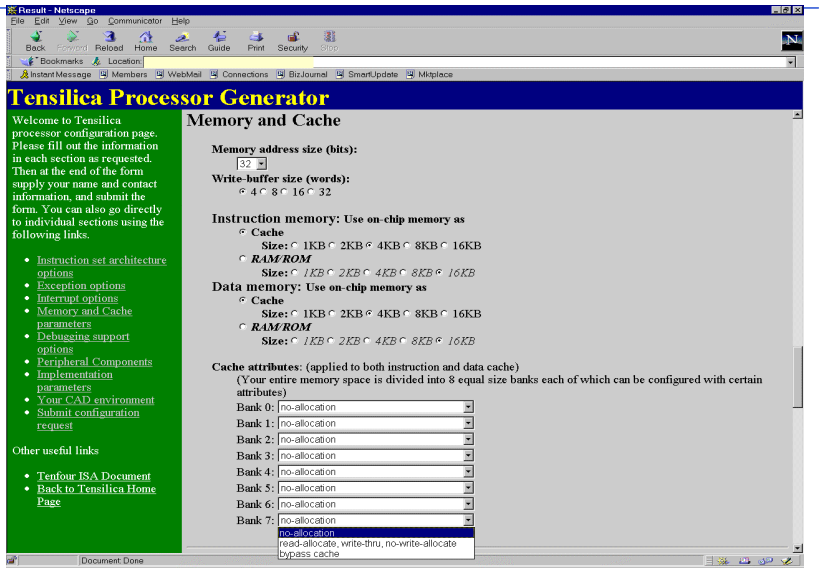
Customized Compiler, Assembler, Linker, Debugger, Simulator

Use a standard cell library to target to the silicon process

Easily select Instruction Set attributes



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?**

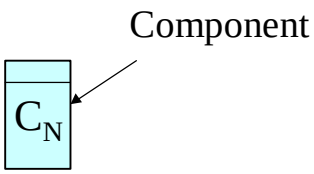
Implementation

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

Verification

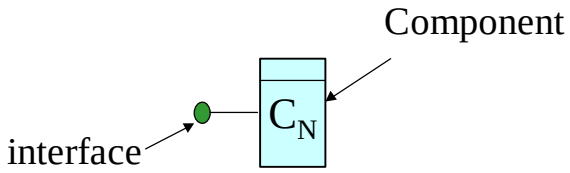
- **How do we verify individual components?**
- **How do we verify component interfaces?**
- **How do we verify a family of parameterizable instances?**

What is a Component?



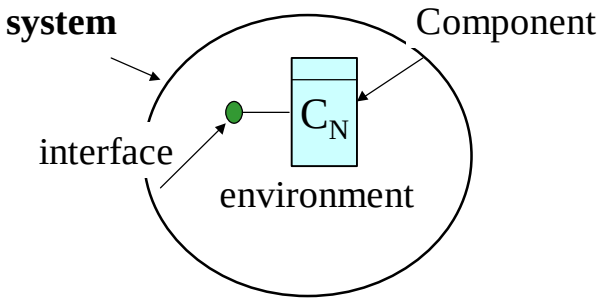
- **A *component* is defined as any fully encapsulated behavior.**
  - **It is analogous to an object in object-oriented programming in that it may have an associated state, behavior, and identity.**
- **We use the term component, rather than object, to stress the fact that the implementation medium– 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?**

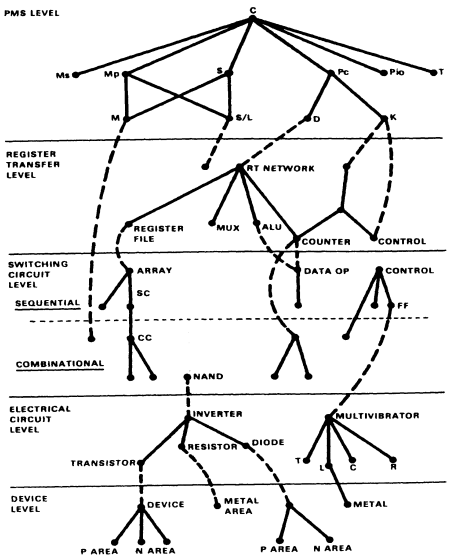
Verification

- **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

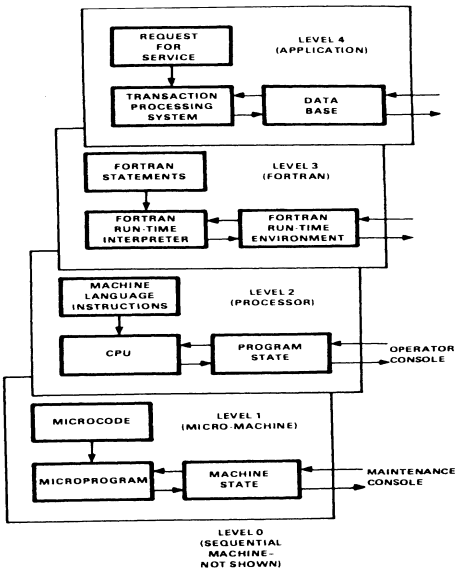
View One:	Structural Levels of a Computer System
View Two:	Levy's Levels of Interpreters
View Three:	Packaging Levels of Integration
View Four:	A Marketplace View of Computer Classes
View Five:	An Applications/Functional View of Computer Classes
View Six:	The Practice of Design
View Seven:	The BLAAUW 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_B Inst_C ... Inst_C may not follow Inst_A	Inst_A Inst_B Inst_C ... 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.

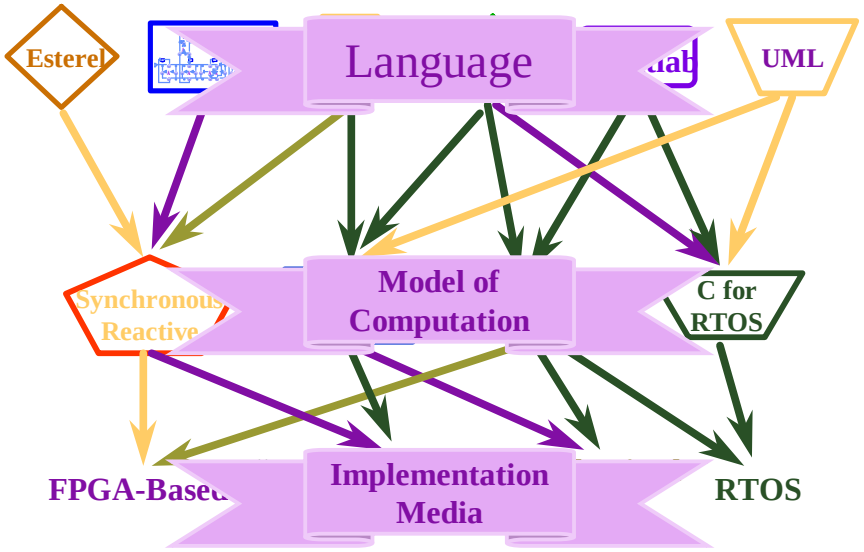
e.g. ∀A { A[i,j] ← 0}

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

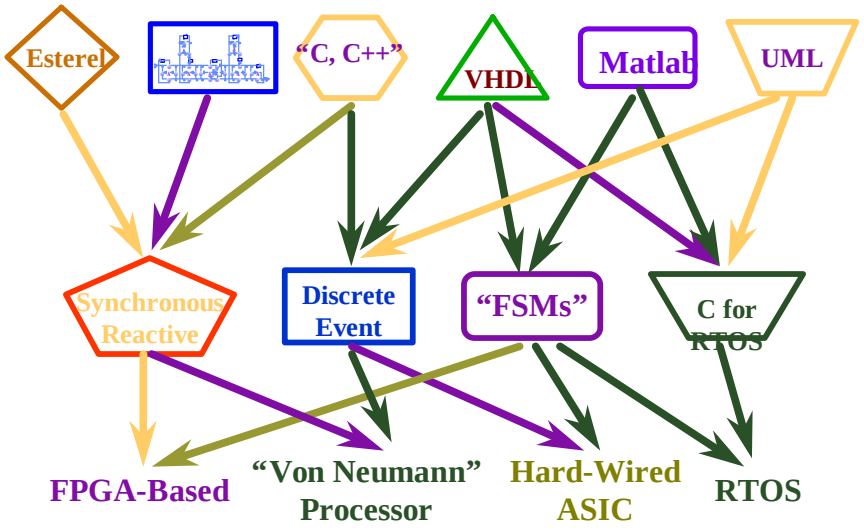
```
e.g. for(i=0; i<N; i++)
         for(j=0; j<M; j++)
             A[i][j] = 0;
```

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

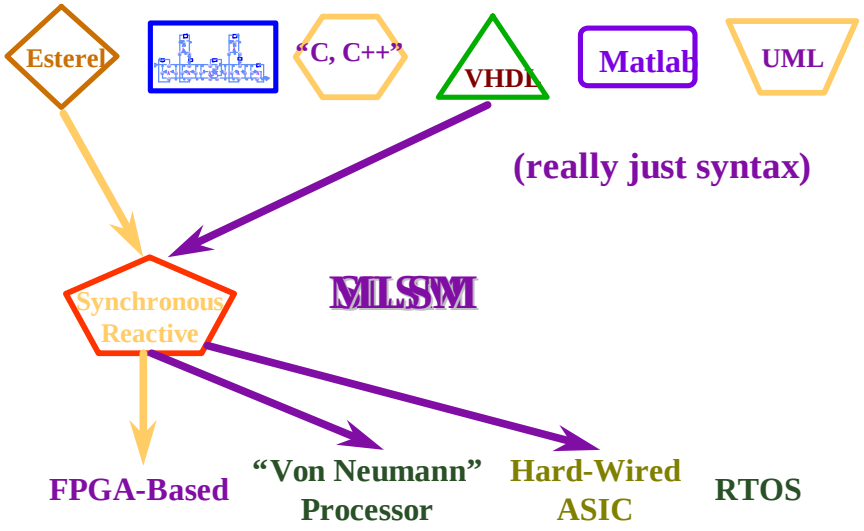
Languages versus Models



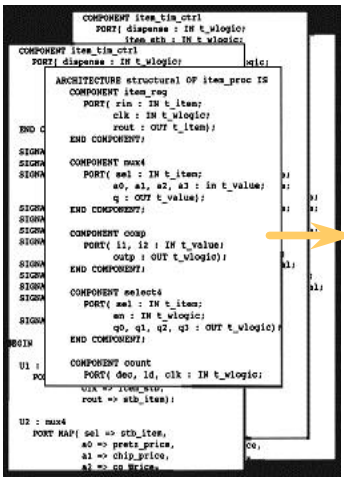
Languages versus Models



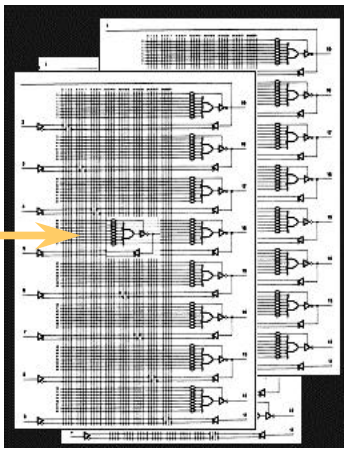
Languages versus Models



VHDL: The “nroff/latex” of Design



VHDL-Based Synthesis System



Encoding Information in Time & Space

```
PROCESS (L(7))
    VARIABLE  V_TRAN_1 : INTEGER;
    VARIABLE  V_DEC_25 : INTEGER;

BEGIN

    IF REGOOO <= V_N THEN
      R_SUM <=  0.0E+00;
      REGOO2 <= 1;
      V_TRAN_1 := REGOOO - 1;
      R_OPT_T <= V_TRAN_1;
      V_DEC_25 := ABS (V_TRAN_1);
      REGOO3 <= V_DEC_25 * 81;
      L(8) <= NOT(L(8));
    ELSE
      IF R_RESID < V_EPSLON THEN
        L(5) <= NOT(L(5));
      ELSE
        REGOO1 <= REGOO1 + 1;
        L(6) <= NOT(L(6));
      END IF;
    END IF;

END PROCESS;
```

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?**

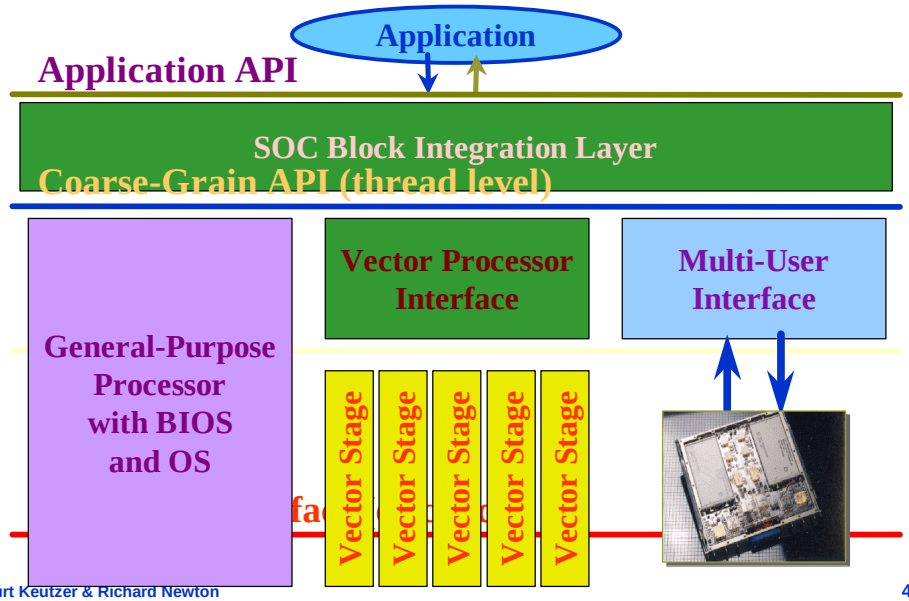
Implementation

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

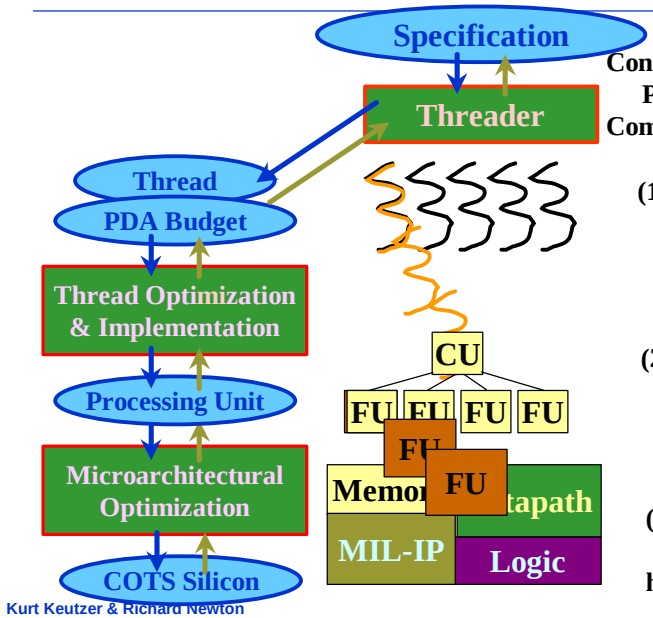
Verification

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

Architecture for SOC



A Vision: Design System 2010



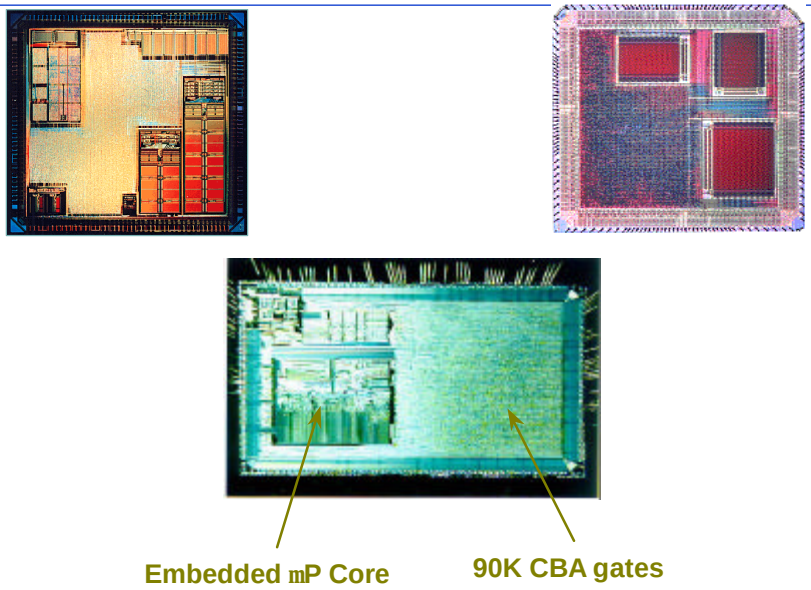
Constraints: Real time, PDA, Precision Communication Complexity

(1)Threads mapped to processing units

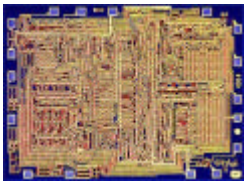
(2) Functional units and coprocessors allocated to processing unit

(3) Functional unit implemented as hardware/"software"

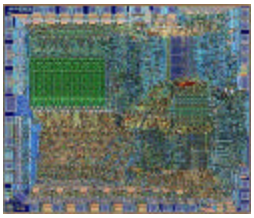
System-On-A-Chip: 1998



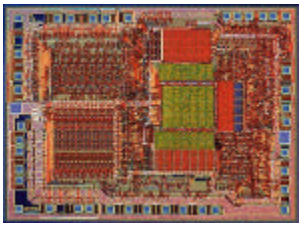
Transition to Extensive Use of Regular Structures



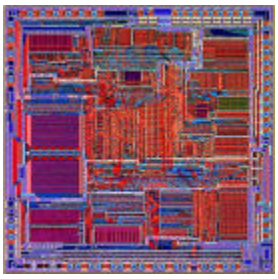
Intel 4004 (‘71)



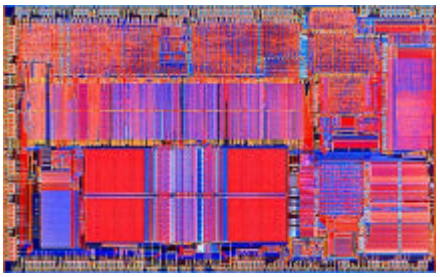
Intel 8080



Intel 8085

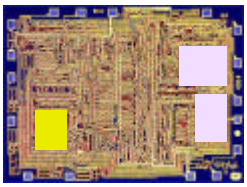


Intel 8286

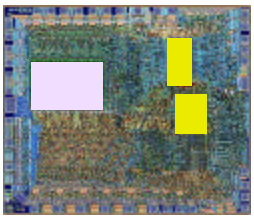


Intel 8486

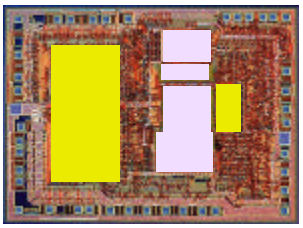
Move Towards Regularity and Programmability



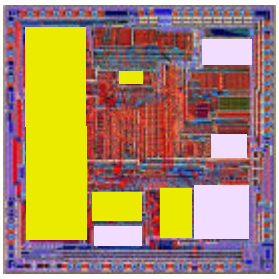
Intel 4004 (‘71)



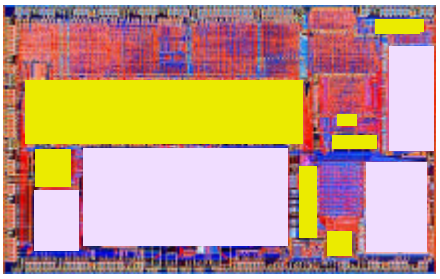
Intel 8080



Intel 8085



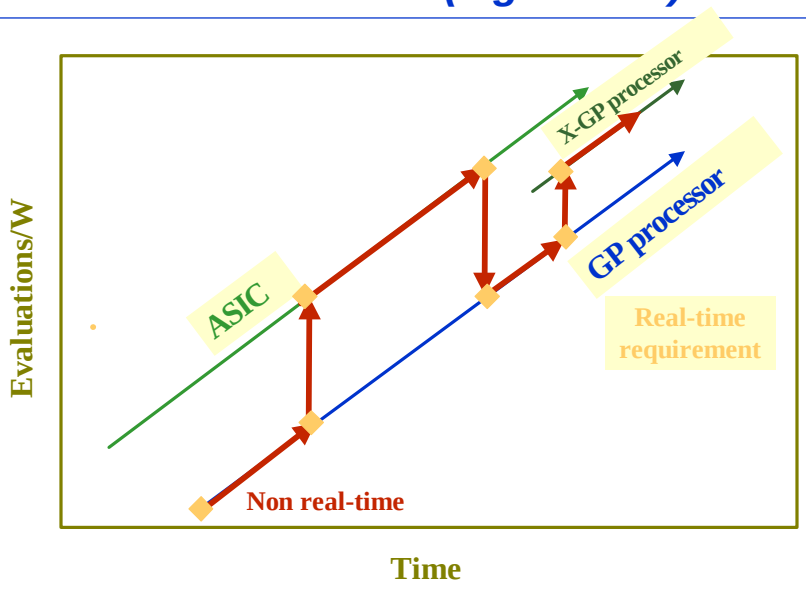
Intel 8286



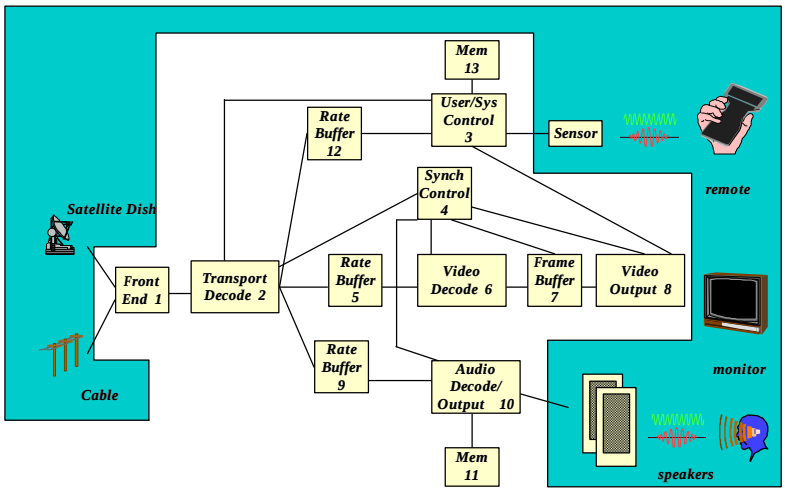
Intel 8486

Regular logic Programmable structure Other (random logic, etc.)

Particular Function (e.g. MPEG)

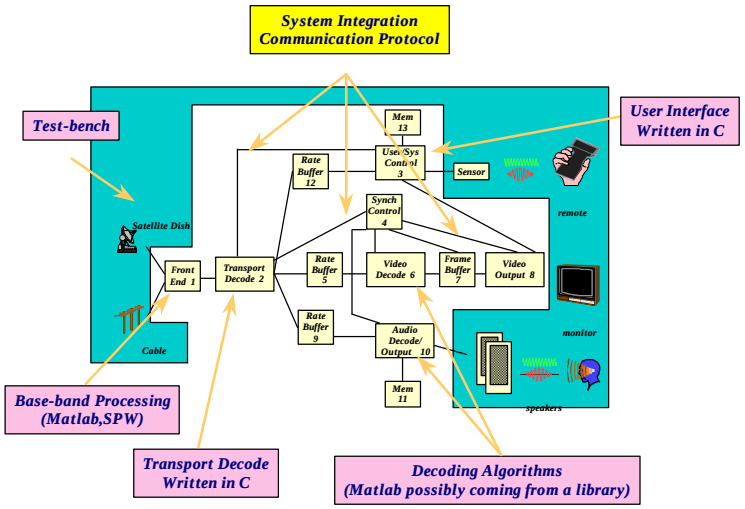


Example of System Behavior



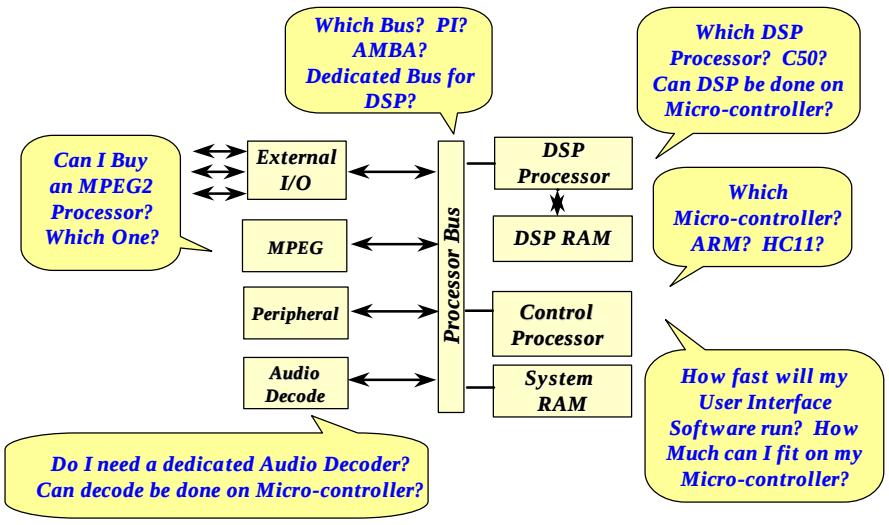
From Alberto Sangiovanni-Vincentelli

IP-Based Design of Behavior



From Alberto Sangiovanni-Vincentelli

IP-Based Design of Implementation



From Alberto Sangiovanni-Vincentelli

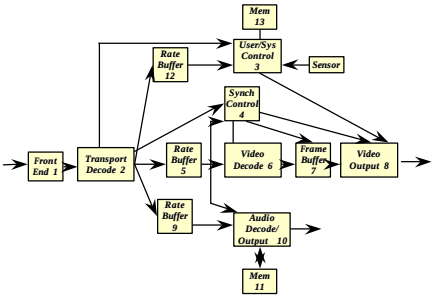
Separate Behavior from Architecture

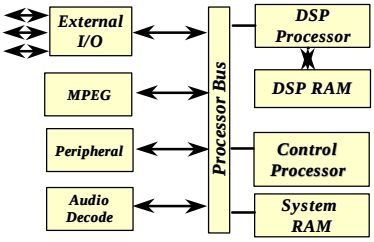
System Behavior

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

Implementation Architecture

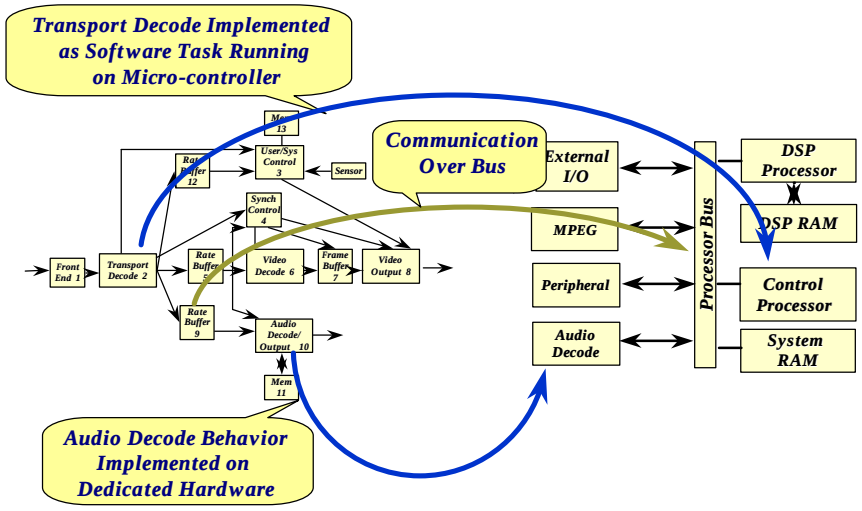
- Hardware and Software
- Optimized Computer





From Alberto Sangiovanni-Vincentelli

Map Between Behavior and Architecture



From Alberto Sangiovanni-Vincentelli

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

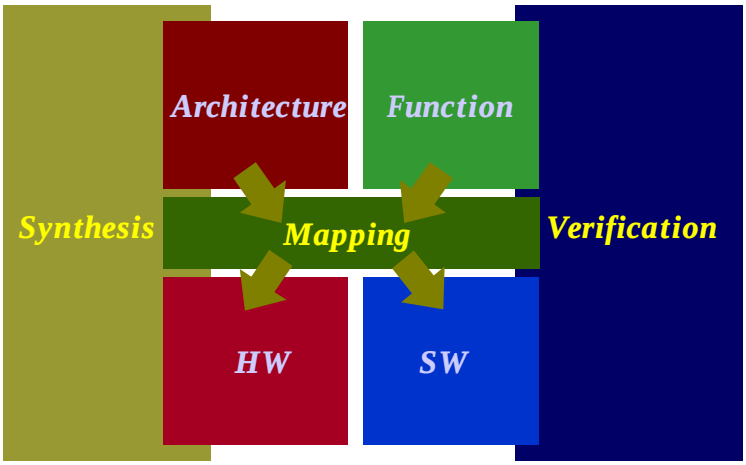
System Level Design

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

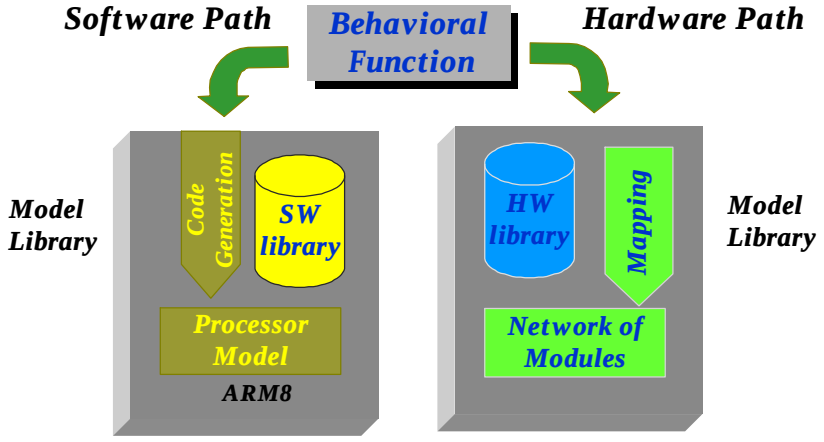
From Alberto Sangiovanni-Vincentelli

System Design



From Alberto Sangiovanni-Vincentelli

Estimation and Modeling



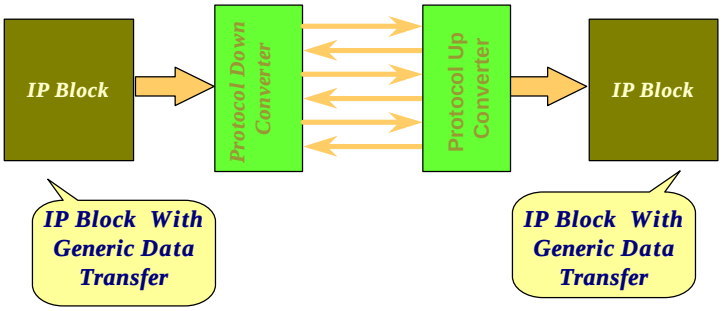
Combines Behavioral Parameters and Architectural Models

From Alberto Sangiovanni-Vincentelli

Communication Refinement

Separate *Function* of blocks from inter-block *Communication*

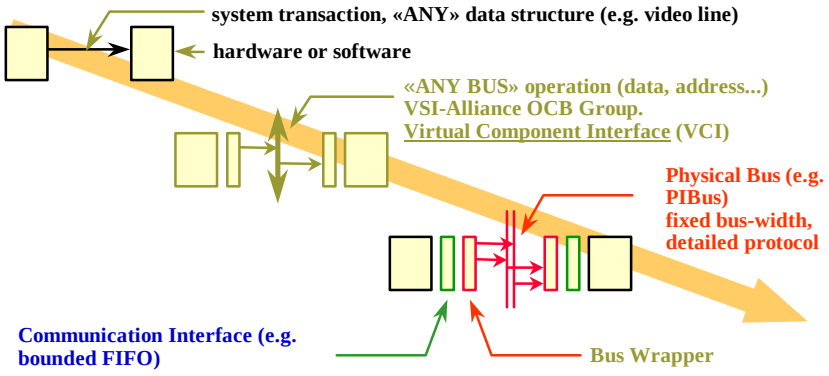
Substitute lower-level detail for communications behavior



From Alberto Sangiovanni-Vincentelli

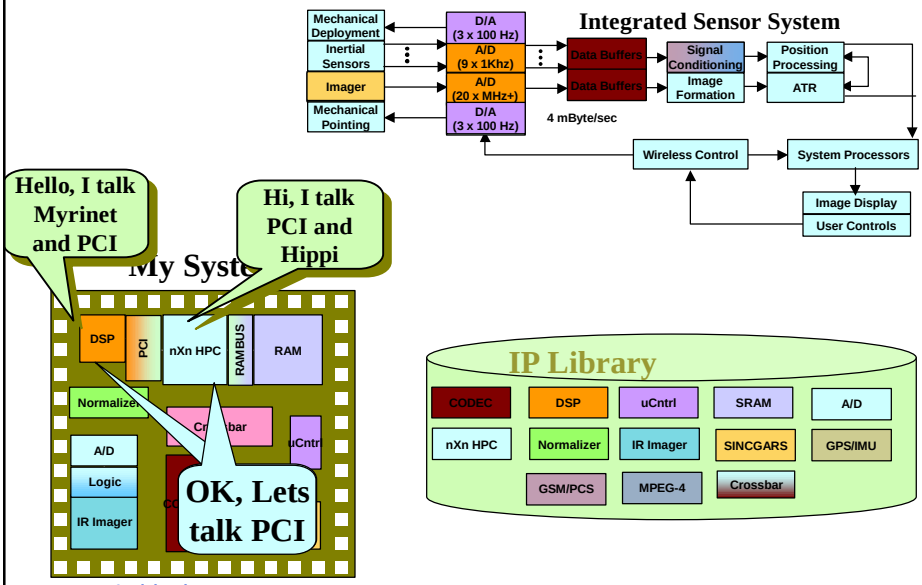
Communication Refinement

Standard interfaces constitute the backbone of an IP market: abstract form the concerns of hardware implementation (multi-target VC), abstract from the concerns of a particular bus (bus-independent VC)



From Alberto Sangiovanni-Vincentelli

IP Integration



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?**

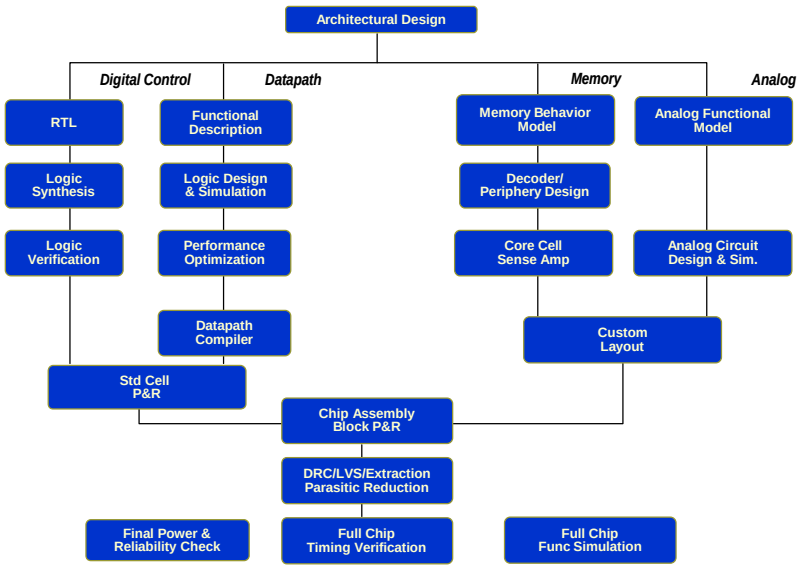
Implementation

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

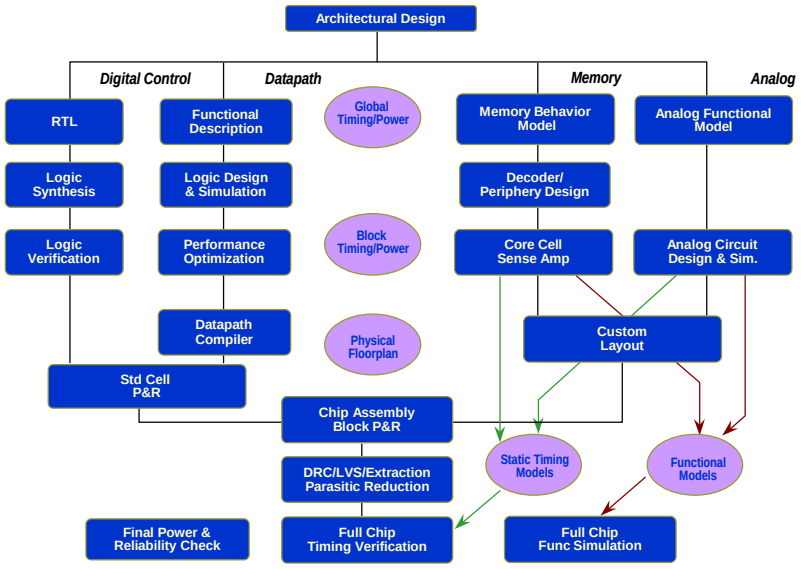
Verification

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

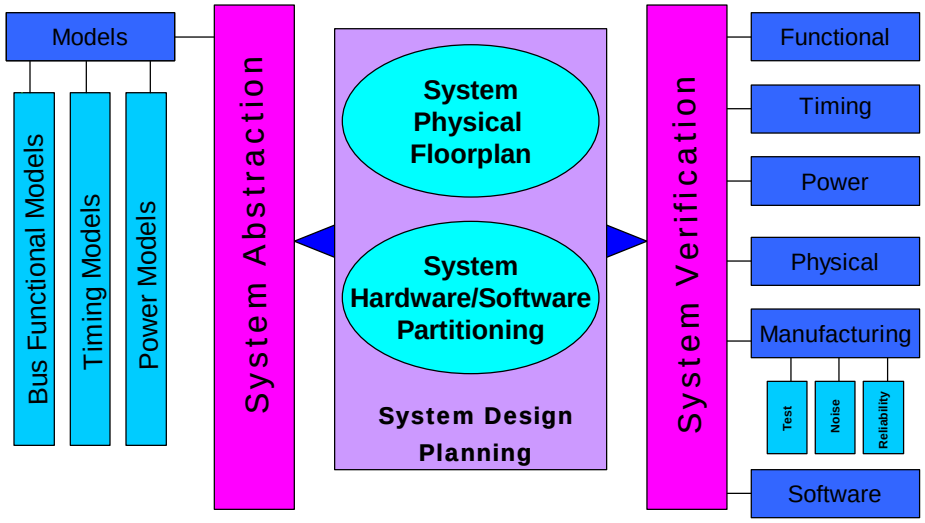
System-on-a-Chip IP Creation and Analysis Flow



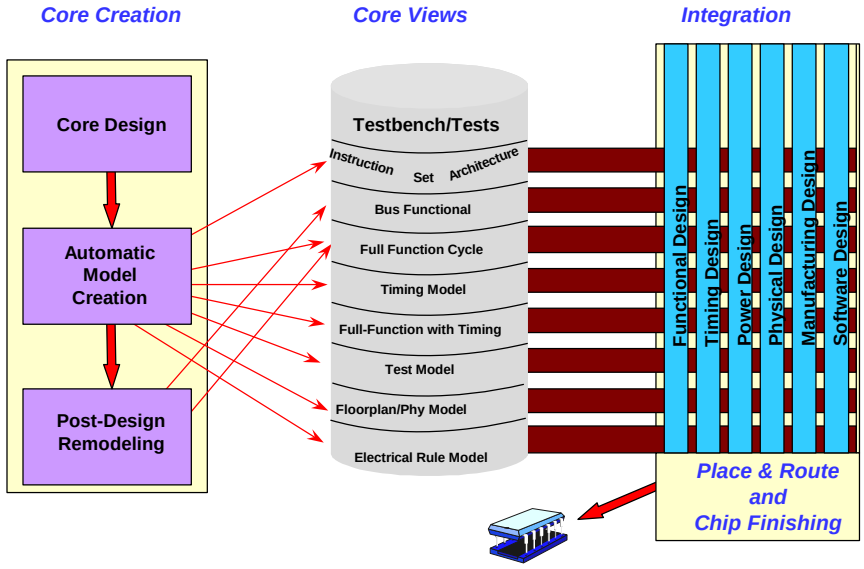
System-on-a-Chip Design Flow



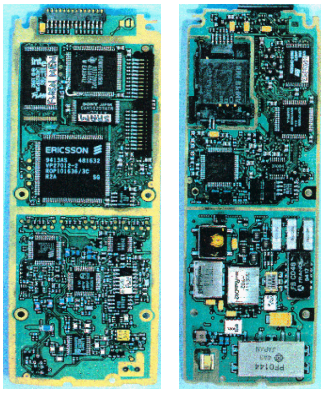
System-on-a-Chip Integration Flow



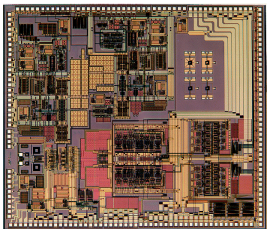
SoC Model Views



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 (description?) 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

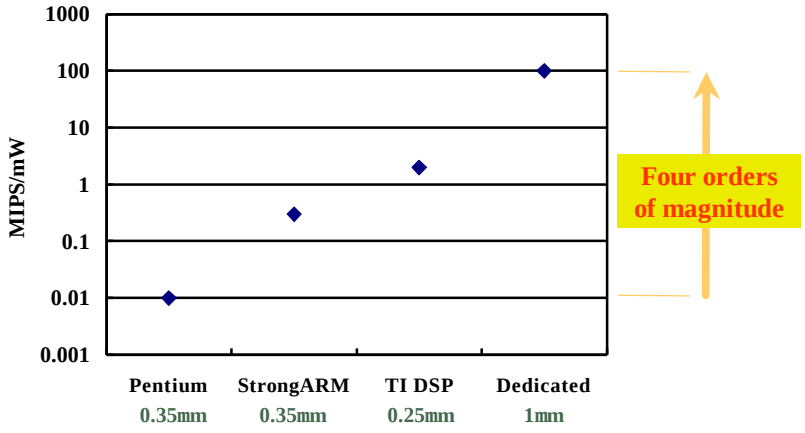
You may work in groups of one or two

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?***