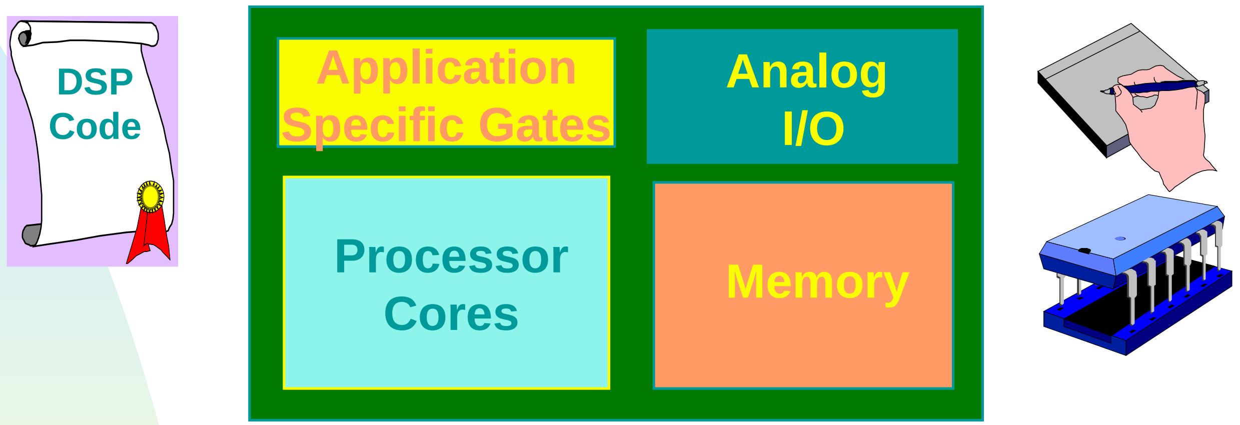
H/W-S/W Co-design

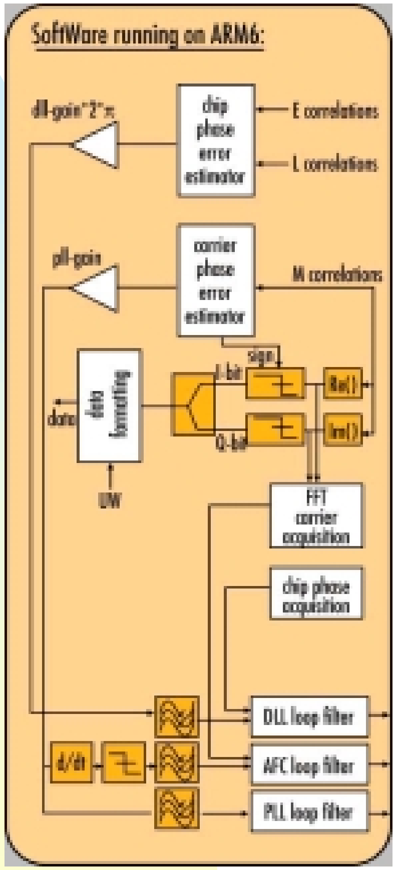
Mani Srivastava
UCLA - EE Department
mbs@janet.ucla.edu

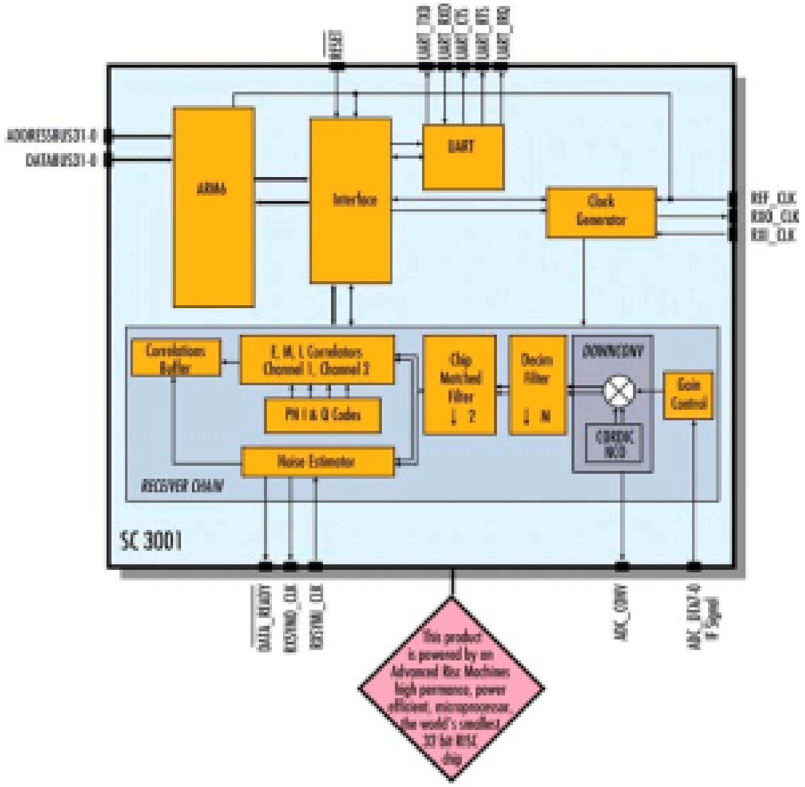
The H/W-/S/W Co-Design Problem



- Embedded systems employ a combination of
  - application-specific h/w (boards, ASICs, FPGAs etc.)
    - performance, low power
  - s/w on prog. processors: DSPs, µcontrollers etc.
    - flexibility, complexity
- Increasingly on the same chip: *System-on-a-chip*

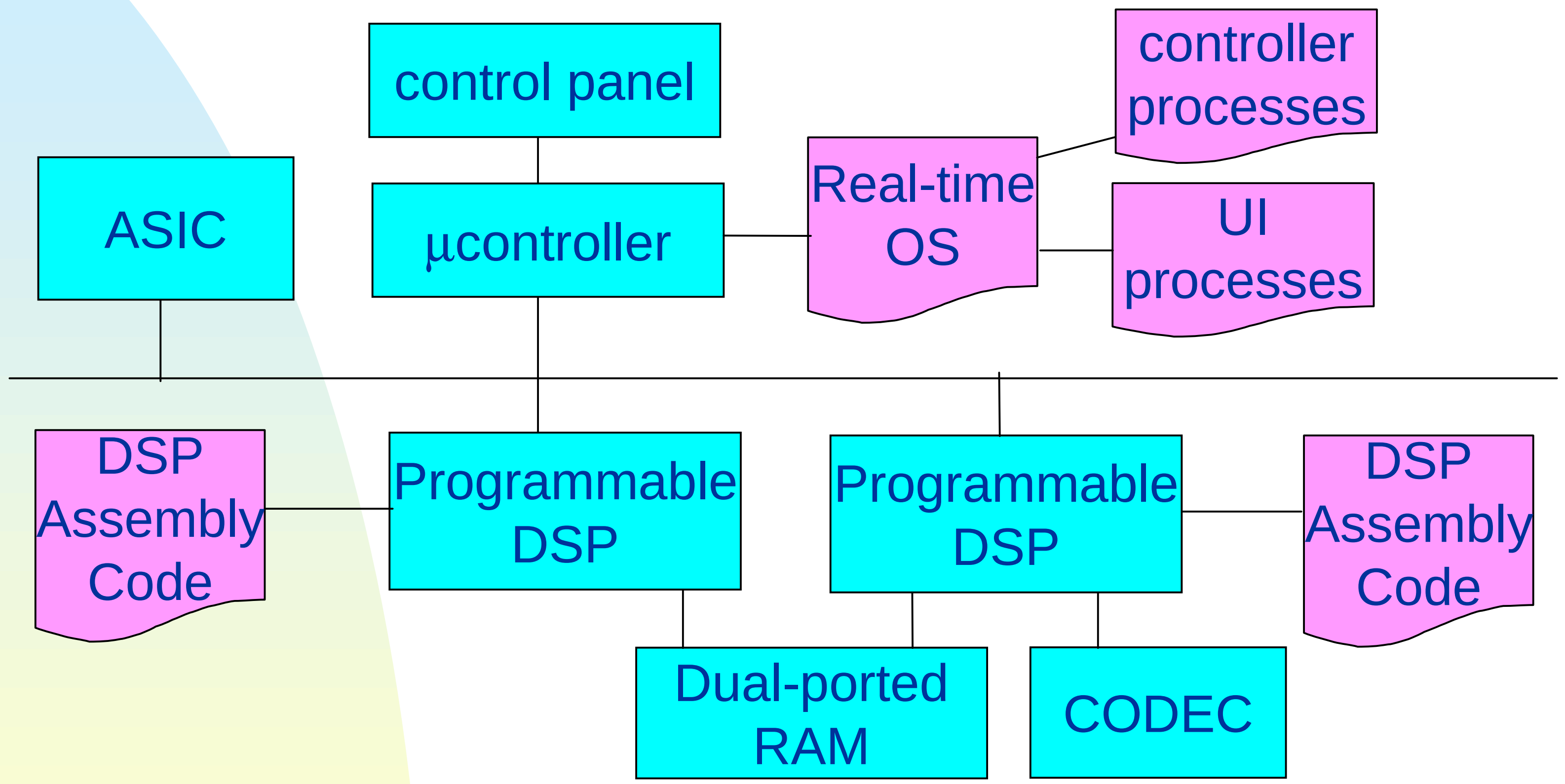
A System on a Chip: a DSSS Receiver





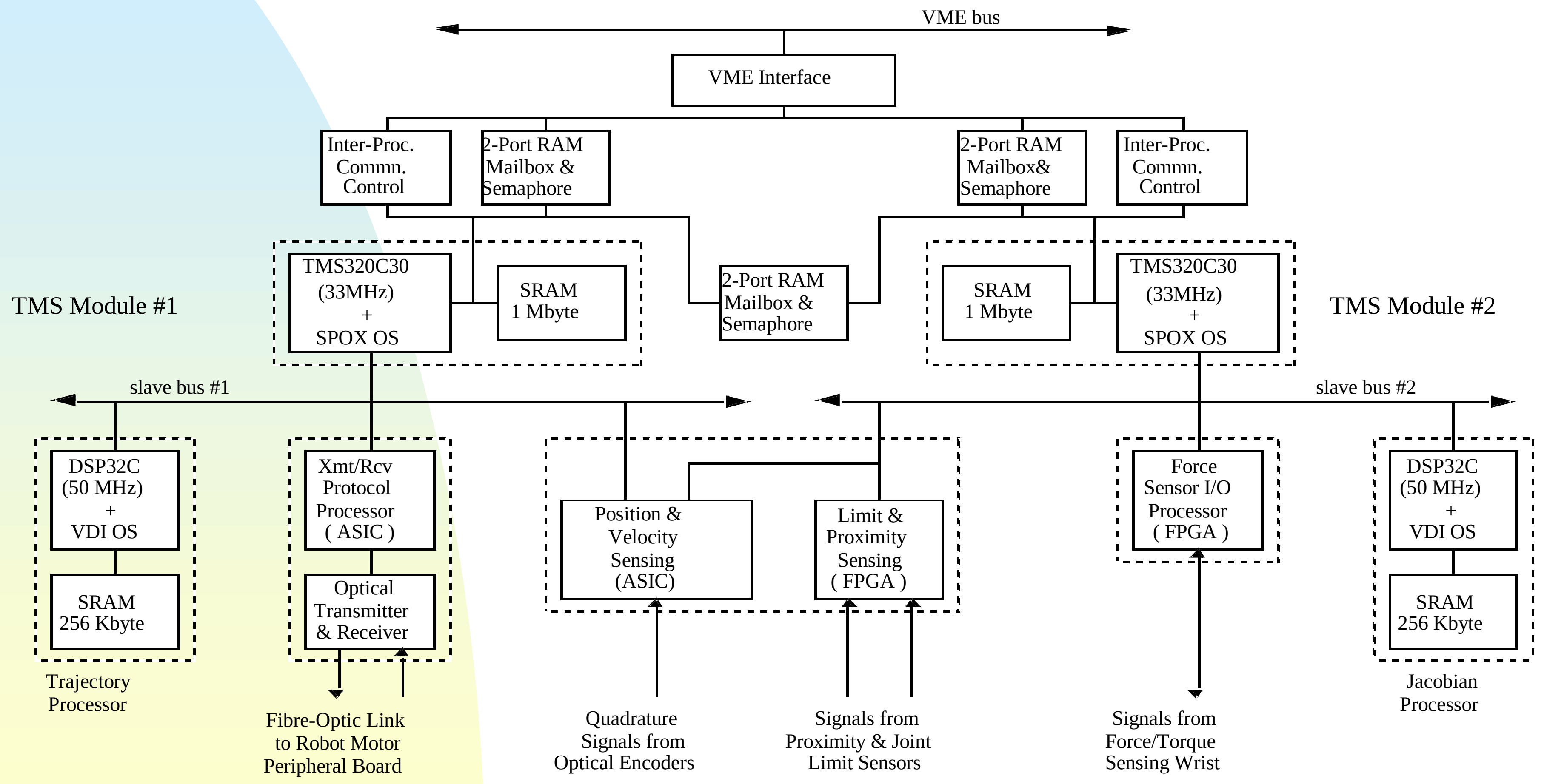
- SC3001 DIRAC chip from Sirius Communications

Complexity and Heterogeneity



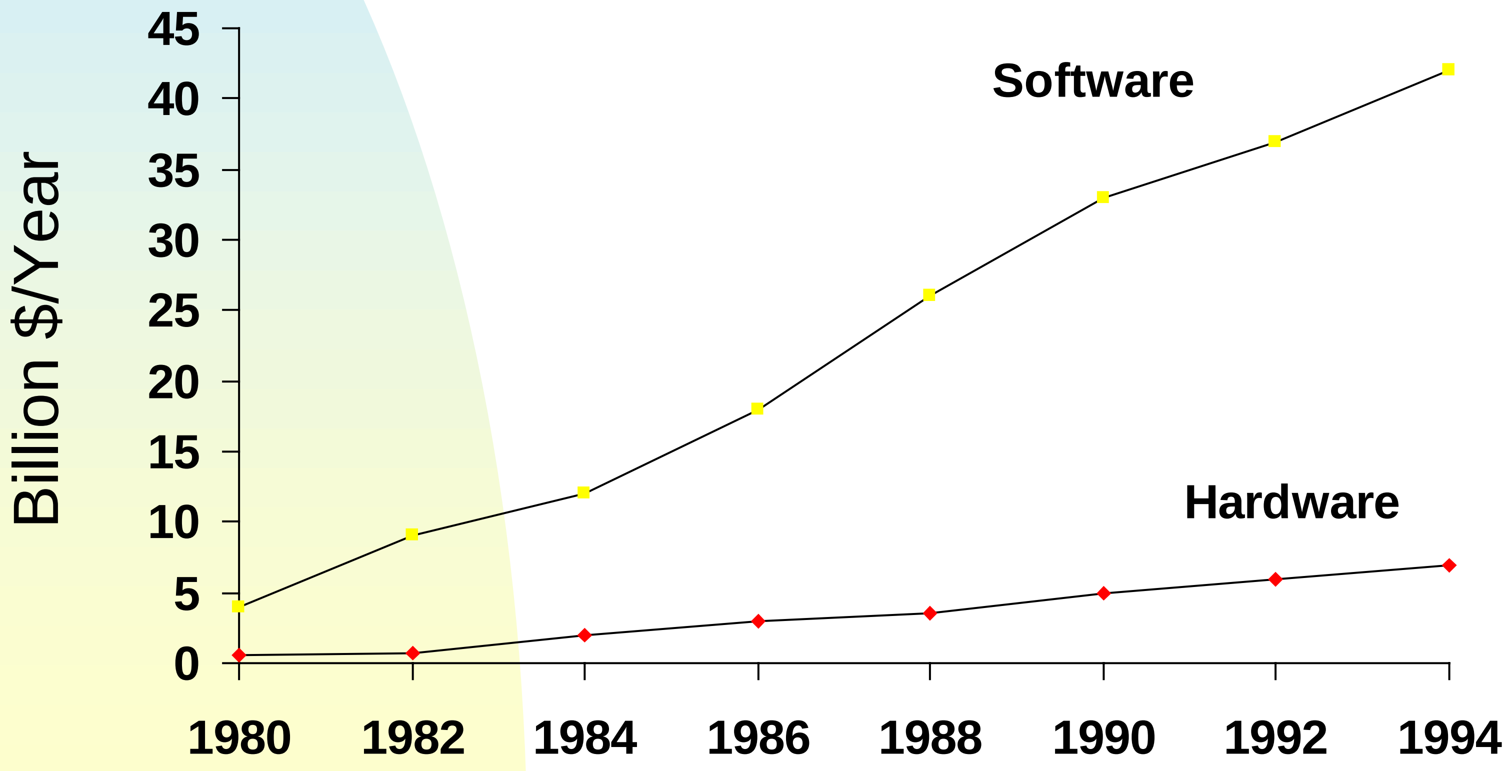
- Heterogeneity within H/W & S/W parts as well
  - S/W: control oriented, DSP oriented
  - H/W: ASICs, COTS ICs

Example: A Robot Controller


VME bus
VME Interface
Inter-Proc. Commn. Control
2-Port RAM Mailbox & Semaphore
2-Port RAM Mailbox& Semaphore
Inter-Proc. Commn. Control
TMS Module #1
TMS320C30 (33MHz) + SPOX OS
SRAM 1 Mbyte
2-Port RAM Mailbox & Semaphore
SRAM 1 Mbyte
TMS320C30 (33MHz) + SPOX OS
TMS Module #2
slave bus #1
slave bus #2
DSP32C (50 MHz) + VDI OS
SRAM 256 Kbyte
Xmt/Rcv Protocol Processor (ASIC)
Optical Transmitter & Receiver
Position & Velocity Sensing (ASIC)
Limit & Proximity Sensing (FPGA)
Force Sensor I/O Processor (FPGA)
DSP32C (50 MHz) + VDI OS
SRAM 256 Kbyte
Trajectory Processor
Fibre-Optic Link to Robot Motor Peripheral Board
Quadrature Signals from Optical Encoders
Signals from Proximity & Joint Limit Sensors
Signals from Force/Torque Sensing Wrist
Jacobian Processor

It is the S/W stupid! Bottleneck not in H/W or ASIC

DoD Embedded System Costs



And, the H/W-S/W Architecture...

- A significant part of the problem is deciding which parts should be in s/w on programmable processors and which in specialized h/w
- Lots of issues in this decision making...

Design of Embedded Systems Today

- Ad hoc approaches based on earlier experience with similar products, & on manual design
- H/W-S/W partitioning decided at the beginning, and then designs proceed separately
- CAD tools take care of h/w fairly well
- But, S/W is a different story…
  - HLLs such as C help, but can't cope with complexity

Holy Grail: H/W-like synthesis & verification from a behavior description of the whole system at a high level of abstraction using formal computation models

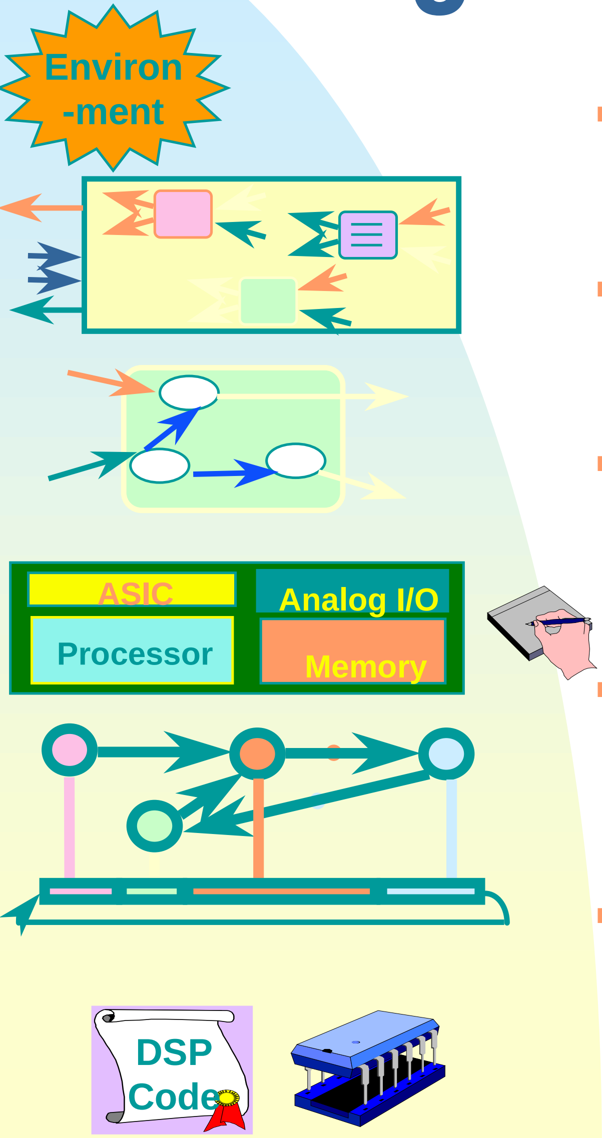
The Holy Grail of Co-design...

- H/W-like synthesis & verification based design methodology
  - system behavior described at a high level of abstraction using formal computation model(s)
  - decomposition into H/W and S/W based on trade-off evaluations from behavioral description
    - H/W & S/W design proceed in parallel with feedback
  - final implementation made as much as possible using automatic synthesis from high level of abstraction
    - ensures “correct by construction” implementations
  - simulation or verfication at higher levels of abstraction

All the "Co-" Buzz Words...

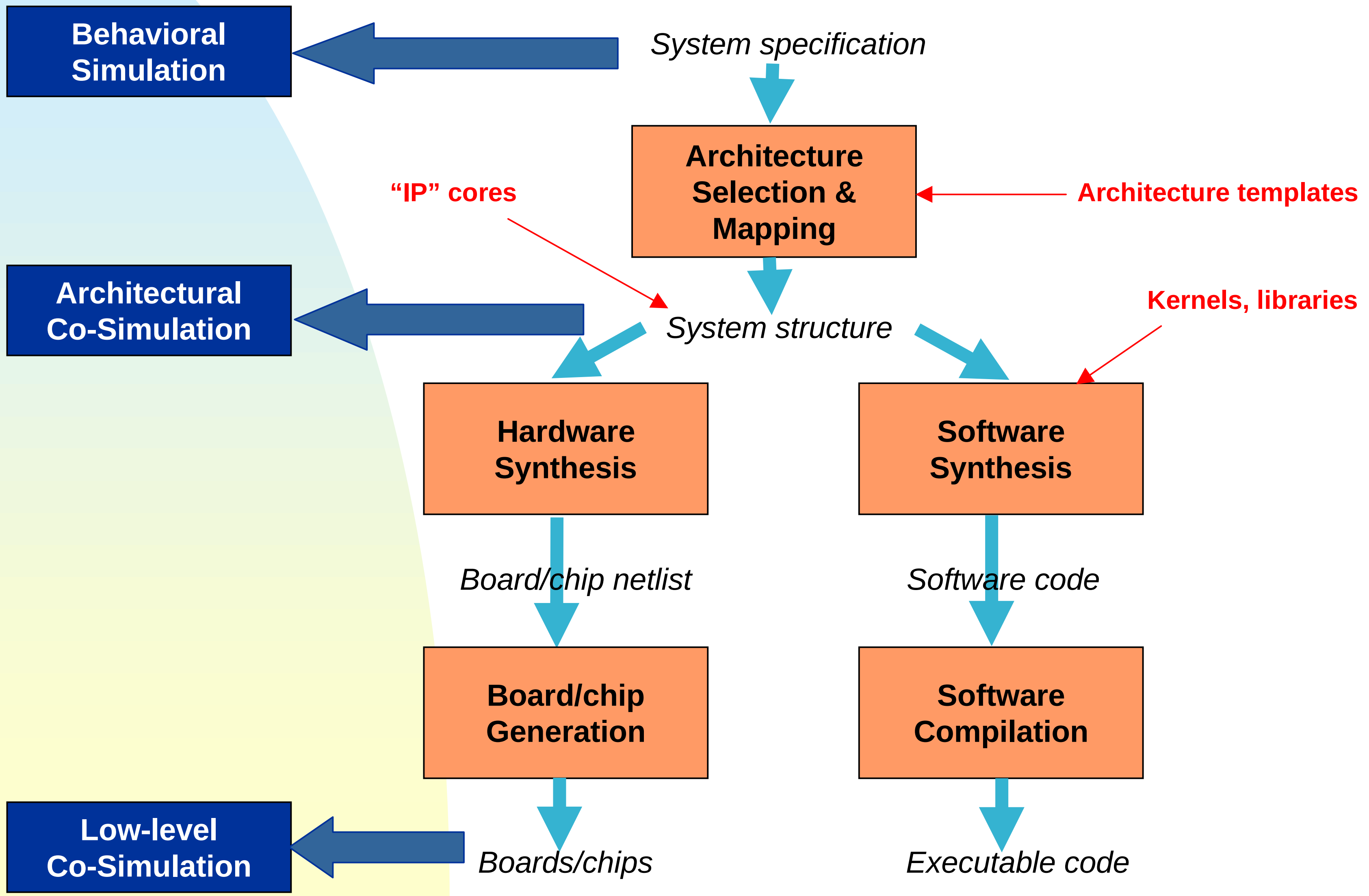
- Co-design
  - joint optimization of hardware and software
- Co-synthesis
  - synthesis assisting co-design
    - mixed h.w-s.w design from (formal) specification
    - rapid exploration of design alternatives
    - enable exploration of architectural alternatives
- Co-simulation
  - simulation of mixed h/w and s/w systems
- Co-specification
  - specifying mixed h/w and s/w systems
- Co-verification

Designing Embedded Systems



- **Modeling**
  - the system to be designed, and experimenting with algorithms involved;
- **Refining (or "partitioning")**
  - the function to be implemented into smaller, interacting pieces;
- **HW-SW partitioning: Allocating**
  - elements in the refined model to either (1) HW units, or (2) SW running on custom hardware or a general microprocessor.
- **Scheduling**
  - the times at which the functions are executed. This is important when several modules in the partition share a single hardware unit.
- **Mapping (Implementing)**
  - a functional description into (1) software that runs on a processor or (2) a collection of custom, semi-custom, or commodity HW.

Typical Co-design Methodology



Some of the Key Problems

- How to model complex/specify systems that will map to hardware and software?
- System level algorithm optimizations
- What are appropriate system architecture models?
- How to partition functions into h/w & s/w?
- How to synthesize s/w? Performance estimation? Impact of OS? Retargetable compilation?
- Synthesis of light-weight app-specific kernels
- What processor to use for s/w? Synthesize?
- How to interface h/w & s/w components?
- How to efficiently simulate h/w & s/w together?

Specification & Modeling

- Design process is a sequence of steps that transform ('refine") a more abstract representation + constraints into a more detailed one
    - "input" representation: specification
    - final representation: implementation
- Representations based on precise mathematical meaning (computation model) are good
    - one can verify and synthesize with guarantees
- But which representation to use for system behavior at high level of abstraction?
    - No single one works well...

What representation for system behavior?

- Many choices… C? DE such as VHDL? SDFG? FSMs? CFSMs? Kahn’s process networks? Hoare’s CSP? Milner’s CCS? Petri nets? Synchronous/reactive languages?
    - But, no single model works well always
- Key issues:
    - complexity or compactness of representation itself
    - ability to naturally express different parts of the system
    - efficient executability

Single Unified Approach

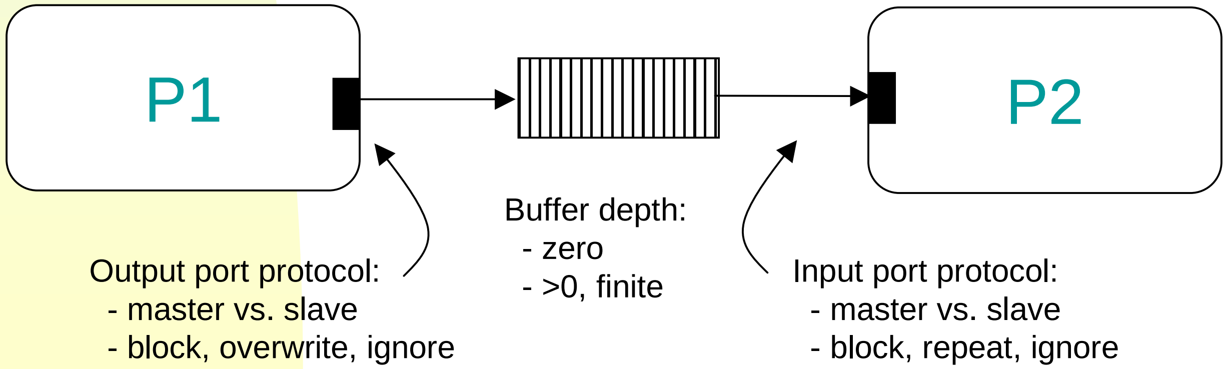
- Example: choose between VHDL and C for a mixed hardware-software design
    - software in VHDL or hardware in C
- Or, bloat VHDL to include a subset for software specification
    - e.g. make ADA a subset of VHDL
    - e.g. enhance C with hardware modeling constructs

Mixed Approach: Heterogeneous Models of Computation

- System viewed as a “composition” of entities whose behavior are described in potentially different models of computation
    - concurrent communication
    - hierarchical containment
- Key problem is defining interaction between fundamentally different models of computation
    - not just a language interfacing issue…
        - e.g. what if two VHDL entities call C procedures that interact?

Network of Processes

- Set of communicating processes with heterogeneous computation models
  - FSMs, sequential processes, SDFG etc.
- Well-defined communication structure
  - e.g. channels connecting read & write ports
    - read protocol, write protocol, message type, buffer depth



Ptolemy’s Approach

- Hierarchical framework
- Specification in one model of computation can contain a primitive that is internally represented in another model of computation
    - “worm holes” connect different domains

Validation

- Process of determining that a design is correct…
    - e.g.checking if there would be deadlock problem
- Three approaches:
    - simulation
        - still the main tool to validate a system model
    - emulation
        - became viable because of reconfigurable hardware platforms, such as Quickturn
    - formal verification
        - easier in case of embedded systems than for, say, generic software programs… e.g. due to finite-state nature etc.

Validation via Simulation

- Main challenge is heterogeneity:
  - both hardware and software components need to be simulated together
- Co-simulation problem!
  - Conflicting requirements
    - execute software as fast as possible, often on a host machine that may be faster than the embedded CPU and usually quite different from it
    - to keep hardware and software simulations synchronized so that they interact just as they would in the real system

Typical Unified Approach to Co-simulation

- General purpose simulator (e.g. VHDL)
  - simulate a model of target CPU executing the S/W
- Different CPU models
  - *gate-level models*
  - *instruction-set architecture models* augmented with hardware interfaces to couple to logic models
  - *bus functional models* of CPU interface - no real program is run, but traces or stochastic traffic used
  - *translation-based models* convert to native code for processor on which simulator is running - need to preserve timing info & coupling to logic models
  - *emulation* used when performance needed

More Distributed Approaches to Co-simulation

- Hardware simulator process loosely links to one or more software processes
- Relative clocks of S/W and H/W simulations need to be synchronized
    - cycle-by-cycle
    - via handshaking
    - S/W is master, H/W is slave
        - S/W simulator sends message to H/W simulator which then either catches up or rolls back (via check-pointing)
    - H/W is master, S/W is slave
        - H/W simulator directly calls communication procedures which in turn call user software code

Validation via Formal Verification

- Mathematically check whether the behavior of a system, described in a formal model, satisfies a given property, also given using a formal model
  - ability to do verification depends on the model of computation (affects decidability, complexity bounds, etc.)
- Specification vs. Implementation Verification
  - former is checking abstract properties of a high level model
  - latter is checking if a low-level model correctly implements a high level model, or satisfies some implementation dependent property

Synthesis

- Refine an abstract specification into a less abstract one
- Combination of three steps:
  - mapping to architecture
    - general structure of implementation is chosen
  - partitioning
    - sections of specifications are bound to architectural units
  - hardware and software synthesis
    - details of units are filled out
    - distinction between synthesis & compilation

Mapping from Specification to Architecture

- Support designer in choosing the right mix of components and implementation technologies
- Input: *functional specification*
  Output: *architecture + assignment of functions to architecture*
- Architecture is generally composed of
  - H/W components (microprocessors, microcontrollers, memories, ASICs, FPGAs etc.)
  - S/W components (device drivers, OS etc.)
  - Interconnect mediua (busses, shared memories etc.)

Mapping from Specification to Architecture (contd.)

- Cost function optimized by mapping process
  - mix of time, area, cost, power etc.
- Current synthesis-based methods impose restriction on target architectures to make mapping problem manageable
  - libraries of pre-defined components
  - no synthesis of memory hierarchy or I/O subsystems based on standard components
  - often communication mechanisms also standardized
    - some work on “interface synthesis”

Partitioning in Embedded Systems

- Interesting because of mix of H/W and S/W
- Four main characteristics of various schemes
    - Specification model supported
        - HDL-based, graph-based
    - Granularity
        - task, operation, operation hierarchy
    - Cost function
        - profiling, synthesis
    - Algorithm
        - greed heuristics, clustering methods, iterative improvement, ILP

Many Partitioning Schemes...

- Examples:
  - all H/W initially, move selected to S/W [Gupta et. al.]
  - all S/W initially, move selected to H/W [Ernst et. al.]
  - others...
- But, no clear winner
  - complex problem, linked to scheduling
  - hard to do an exact formulation with realistic cost estimate of communication overhead etc.
  - many people believe that this is best done manually
    - Berkeley's POLIS, yours truly...

Hardware & Software Synthesis

- Realize specification with minimum cost
  - given specification, architecture, and mapping
- Done after partitioning, usually
  - sometime before partitioning to provide cost estimates
- Typically S/W is assumed to run on off-shelf processors
  - de-couples H/W and S/W design problems
  - also, lower cost
  - but, some allow simultaneous design of S/W and processor it will run on...

Application-Specific Instruction Processors (ASIPs)

- Processors with instruction-sets tailored to specific applications or application domains
    - instruction-set generation as part of synthesis
- Pluses:
    - customization yields lower area, power etc. while r
- Minuses:
    - higher h/w & s/w development overhead
        - design, compilers, debuggers
        - higher time to market

ASIPs vs. General Purpose Processors

- Important issues in GPPs:
  - backward compatibility
  - compiler support
  - optimal performance on wide variety of apps
- Important issue in embedded systems
  - addition of new functionality in future
  - user interaction
  - satisfying timing constraints
- ASIPs are a compromise between ASICs & GPPs
  - FPGAs are another option!

The Old and the New

- H/W synthesis for application-specific hardware (on ASICs, FPGAs etc.)
  - classical high-level synthesis problem
    - Miodrag's recent lectures...
- Software synthesis
  - relatively new problem
  - VERY hard for general-purpose computing
    - failures led to suspicion
  - but easier in embedded systems
    - S/W is constrained due to real-time & physical issues
    - often no virtual memory, no dynamic task creation or memory allocation, even no stack etc.

"CAD" Methodology for Software in Embedded Systems

- Current practices are ad-hoc
  - art form: hand-tuned implementations by gurus
  - "reworking" and "debugging" is hard
  - often "single task" with interrupt handlers
  - reuse (object technologies) helpful but...
- Software synthesis
  - software generation from abstract models by using CAD tools that do optimizations for power, size etc.
  - code generation: retargetable compilation, processor descriptions, under timing constraints
  - performance estimation: power, time etc.

Issues in Software Synthesis

- Specification formalism
- Interfacing mechanisms (to S/W and to H/W)
  - none, synthesized, device-driver synthesis
- Constraint granularity
  - task, operation
- When is scheduling done
  - static, quasi-static, dynamic etc.
- Scheduling method
  - RMA, EDF, DMA etc.
  - domain-specific scheduling algorithms

Other Issues...

- Software Analysis: estimate “cost” of S/W
  - time
    - techniques applied from H/W CAD to detect false paths
  - power estimation
- Optimal layout of data structures
- Interfacing H/W and S/W
  - device driver and interface logic synthesis
- Processors with peripheral devices