HW/SW Co-Design

Outline

- Introduction
- When to Use Accelerators
  - Real Time Scheduling
- Accelerated System Design
  - Architecture Selection
  - Partitioning and Scheduling
- Key Recent Trends

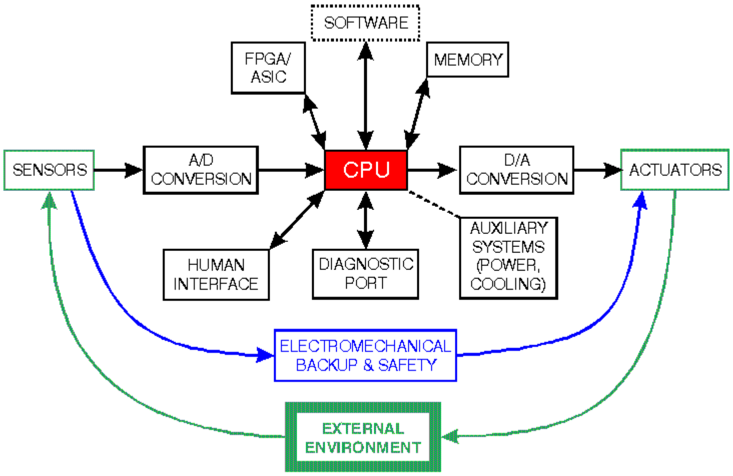
Embedded Systems

- Signal processing systems
  - radar, sonar, real-time video, set-top boxes, DVD players, medical equipment, residential gateways
- Mission critical systems
  - avionics, space-craft control, nuclear plant control
- Distributed control
  - network routers & switches, mass transit systems, elevators in large buildings
- “Small” systems
  - cellular phones, pagers, home appliances, toys, smart cards, MP3 players, PDAs, digital cameras and camcorders, sensors, smart badges

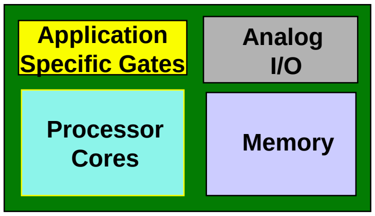
Typical Characteristics of Embedded Systems

- Part of a larger system
  - not a “computer with keyboard, display, etc.”
- HW & SW do application-specific function – not G.P.
  - application is known a priori
  - but definition and development concurrent
- Some degree of re-programmability is essential
  - flexibility in upgrading, bug fixing, product differentiation, product customization
- Interact (sense, manipulate, communicate) with the external world
- Never terminate (ideally)
- **Operation is time constrained: latency, throughput**
- **Other constraints: power, size, weight, heat, reliability etc.**
- **Increasingly high-performance (DSP) & networked**

"Traditional" Software Embedded Systems = CPU + RTOS



Modern Embedded Systems?

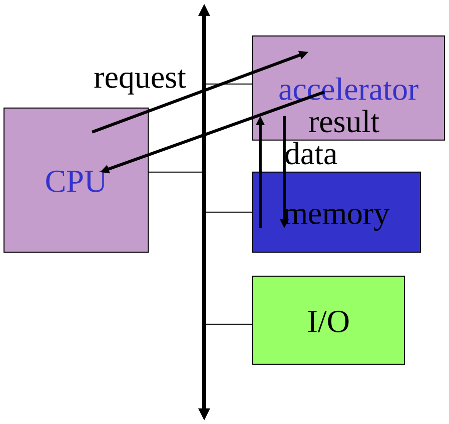


- Embedded systems employ a combination of
  - **application-specific h/w** (boards, ASICs, FPGAs etc.)
    - ❄ performance, low power
  - **s/w on prog. processors**: DSPs, µcontrollers etc.
    - ❄ flexibility, complexity
  - mechanical transducers and actuators

Accelerating Systems

- Use additional computational unit(s) dedicated to some functions
  - Hardwired logic.
  - Extra CPU.
- Hardware/Software Co-design: joint design of hardware and software architectures.
  - performance analysis
  - scheduling and allocation

Accelerated System Architecture



Accelerator vs. Co-Processor

- A co-processor executes instructions.
  - Instructions are dispatched by the CPU.
- An accelerator appears as a device on the bus.
  - The accelerator is controlled via registers.

Accelerator Implementations

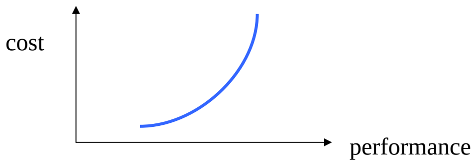
- Application-specific integrated circuit.
- Field-programmable gate array (FPGA).
- Standard component.
  - Example: graphics processor.
- SoCs enable multiple accelerators, CPUs, peripherals, and some memory to be placed within a single chip.

System Design Tasks

- Design a heterogeneous multiprocessor architecture that satisfies the design requirements.
  - Processing element (PE): CPU, accelerator, etc.
- Program the system.

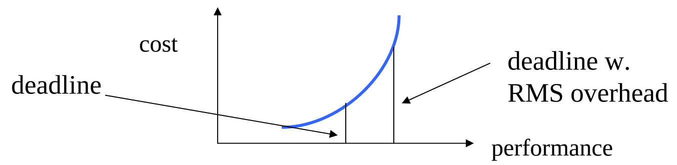
Why Accelerators?

- Better cost/performance.
  - Custom logic may be able to perform operation faster or at lower power than a CPU of equivalent cost.
  - CPU cost is a non-linear function of performance.



Why Accelerators? cont’d.

- Better real-time performance.
  - Put time-critical functions on less-loaded processing elements.
  - Rate Monotonic Scheduling (RMS) utilization is ‘*limited*’---extra CPU cycles must be reserved to meet deadlines. (*see next section*)



Why Accelerators? cont’d.

- Good for processing I/O in real-time.
- May consume less energy.
- May be better at streaming data.
- **May not be able to do all the work on even the largest single CPU…**

Outline

- Introduction
- When to Use Accelerators
  - ➡ Real Time Scheduling
- Accelerated System Design
  - Architecture Selection
  - Partitioning and Scheduling
- Key Recent Trends

Real Time Scheduling

- Scheduling Policies
  - RMS – Rate Monotonic Scheduling:
    - ❄ Task Priority = Rate = 1/Period
    - ❄ RMS is the optimal preemptive *fixed-priority* scheduling policy.
  - EDF – Earliest Deadline First:
    - ❄ Task Priority = Current Absolute Deadline
    - ❄ EDF is the optimal preemptive *dynamic-priority* scheduling policy.

Real Time Scheduling Assumptions

- Scheduling Assumptions
  - Single Processor
  - All Tasks are Periodic
  - Zero Context-Switch Time
  - Worst-Case Task Execution Times are Known
  - No Data Dependencies Among Tasks.
- RMS and EDF have both been extended to relax these assumptions.

Metrics

- How do we evaluate a scheduling policy:
  - Ability to satisfy all deadlines.
  - CPU utilization---percentage of time devoted to useful work.
  - Scheduling overhead---time required to make scheduling decision.

Rate Monotonic Scheduling

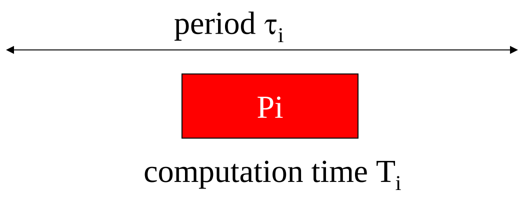
- RMS (Liu and Layland): widely-used, analyzable scheduling policy.
- Analysis is known as Rate Monotonic Analysis (RMA).

RMA model

- All process run on single CPU.
- Zero context switch time.
- No data dependencies between processes.
- Process execution time is constant.
- Deadline is at end of period.
- **Highest-priority ready process runs.**

Process Parameters

- T_i is execution time of process *i*; τ_i is period of process *i*.



period τ_i

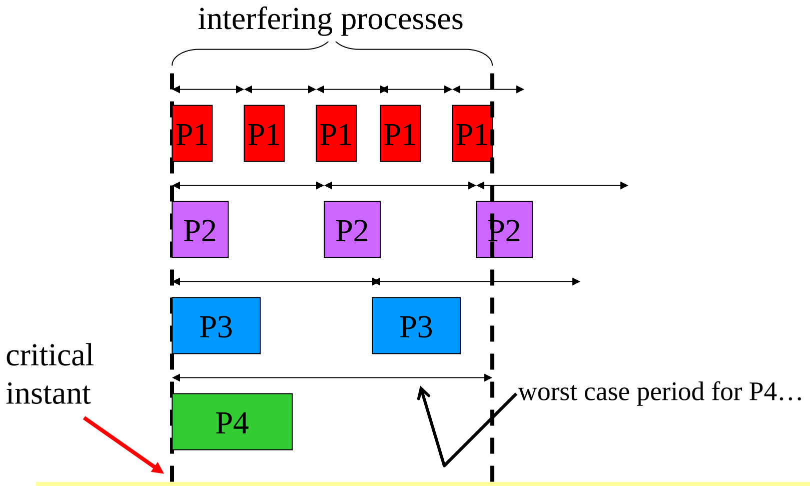
Pi

computation time T_i

Rate-Monotonic Analysis

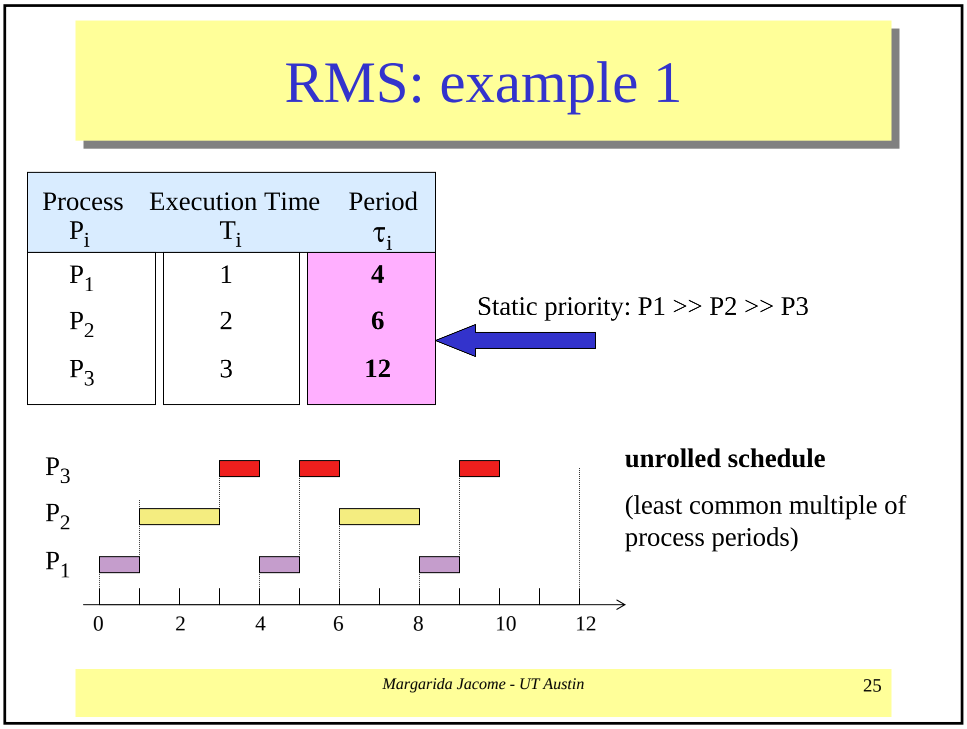
- Response time: time required to finish a process/task.
- Critical instant: scheduling state that gives worst response time.
  - Critical instant occurs when all higher-priority processes are ready to execute.

Critical Instant



RMS priorities

- Optimal (fixed) priority assignment:
  - shortest-period process gets highest priority;
    - priority based preemption can be used…
  - priority inversely proportional to period;
  - break ties arbitrarily.
- No fixed-priority scheme does better.
  - *RMS provides the highest worst case CPU utilization while ensuring that all processes meet their deadlines*



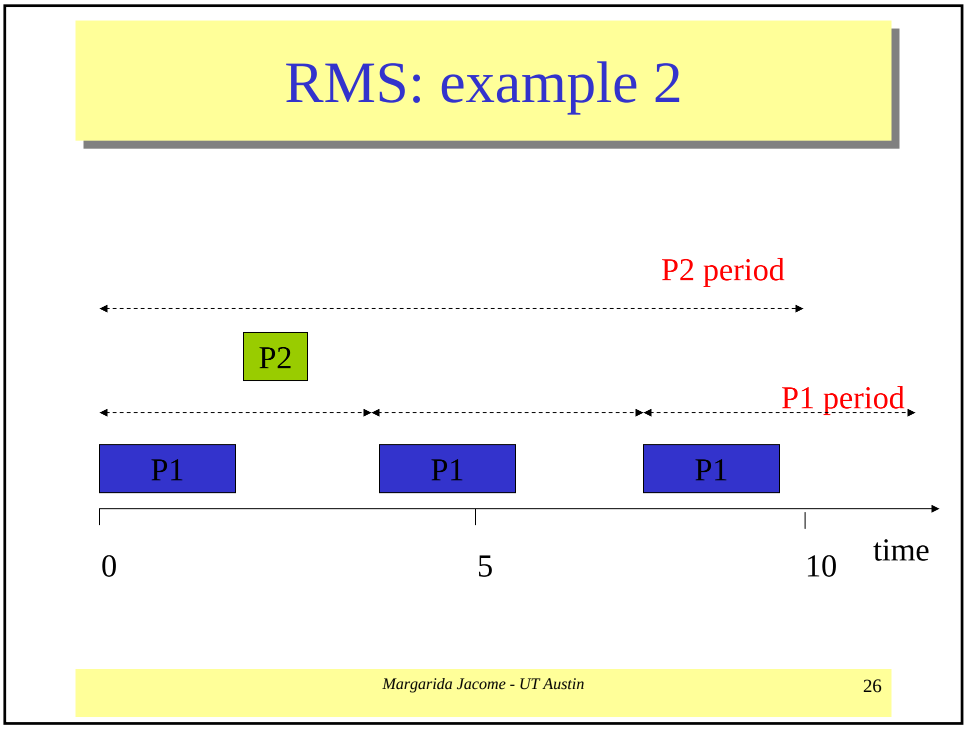
RMS: example 1

Process P_i	Execution Time T_i	Period τ_i
P_1	1	**4**
P_2	2	**6**
P_3	3	**12**

Static priority: P1 >> P2 >> P3

unrolled schedule

(least common multiple of process periods)



RMS: example 2

RMS CPU utilization

- Utilization for *n* processes is
  - $\sum_i T_i / \tau_i$
- As number of tasks approaches infinity, the **worst case** maximum utilization approaches 69%.
  - Yet, is not uncommon to find total utilizations around .90 or more (.69 is worst case behavior of algorithm)
  - Achievable utilization is strongly dependent upon the relative values of the periods of the tasks comprising the task set…

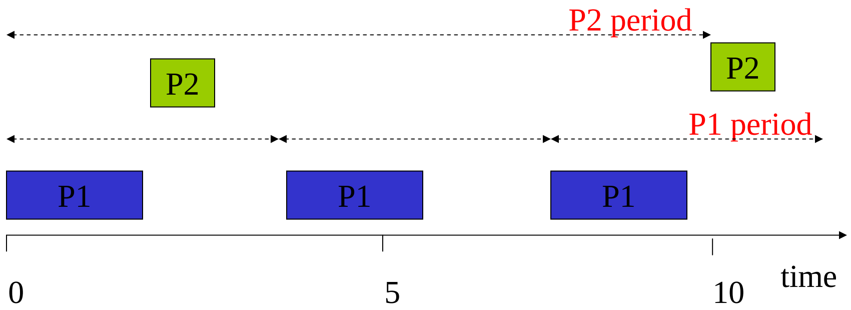
RMS: example 3

Process P_i	Execution Time T_i	Period τ_i
P_1	1	**4**
P_2	6	**8**

Is this task set schedulable?? If yes, give the CPU utilization.

RMS CPU utilization, cont'd.

- RMS cannot asymptotically **guarantee** use of 100% of CPU, even with zero context switch overhead.
  - Must keep idle cycles available to handle worst-case scenario.
- However, RMS guarantees all processes will always meet their deadlines.



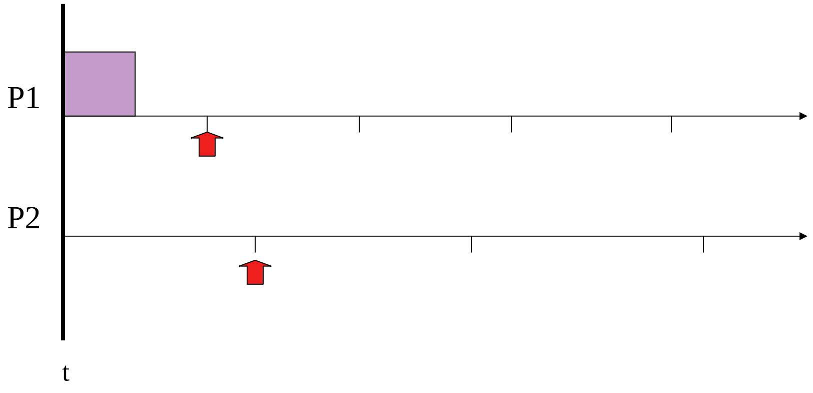
RMS implementation

- Efficient implementation:
  - scan processes;
  - choose highest-priority active process.

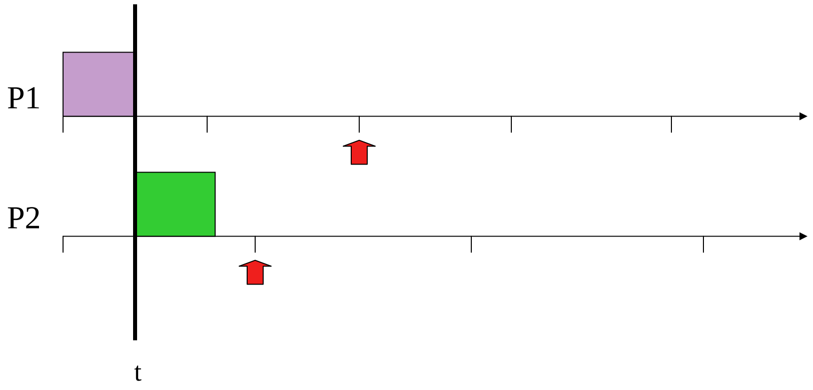
Earliest-deadline-first scheduling

- EDF: **dynamic** priority scheduling scheme.
- Process closest to its deadline has highest priority.
- Requires recalculating processes at every timer interrupt.

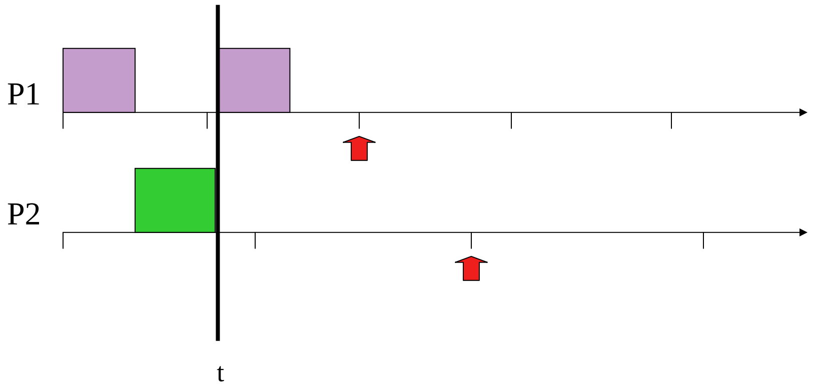
EDF example



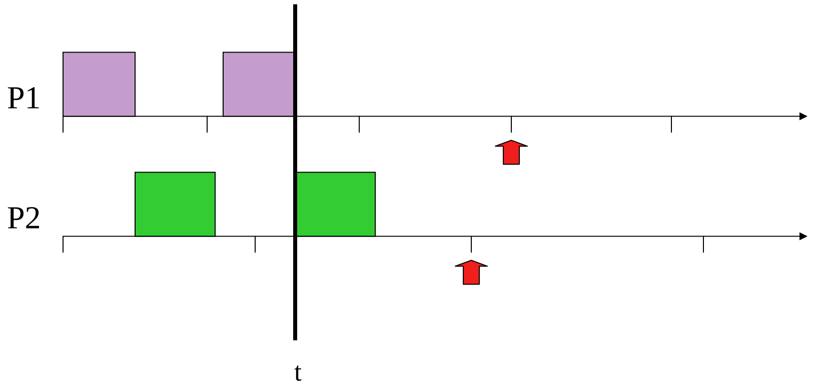
EDF example



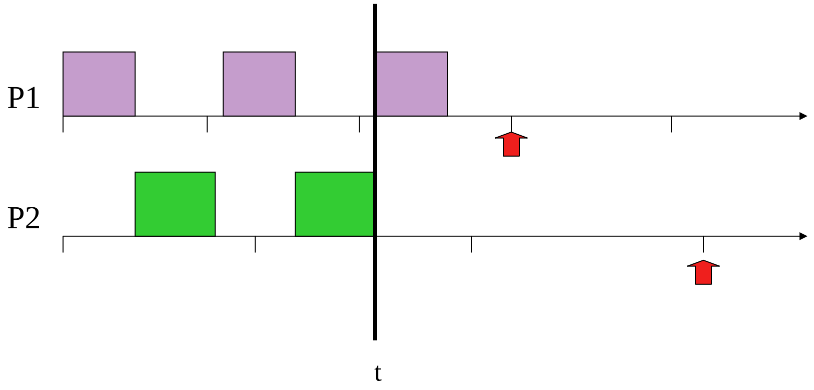
EDF example



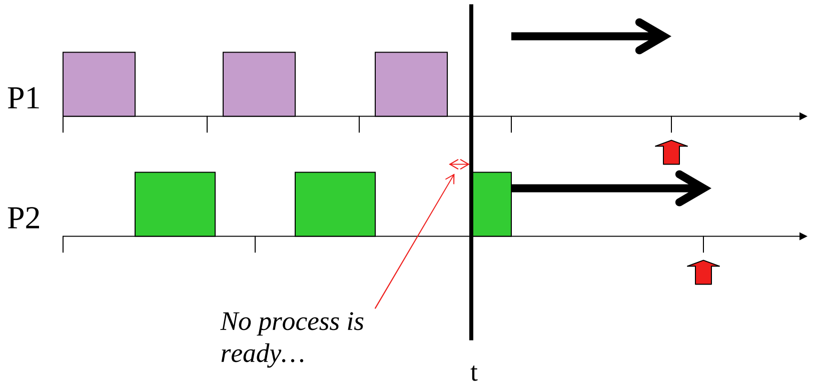
EDF example



EDF example

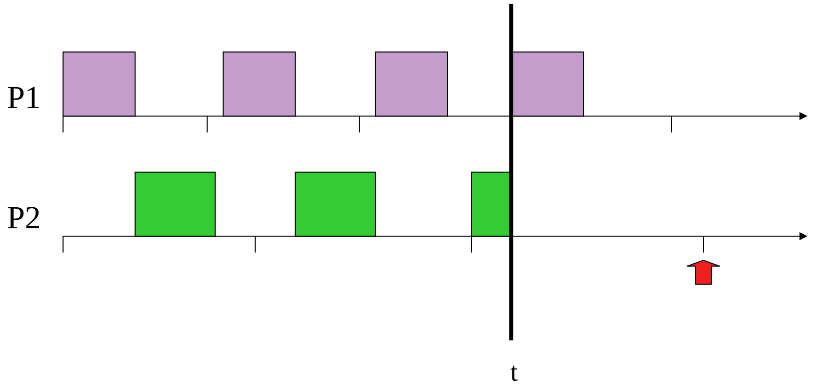


EDF example

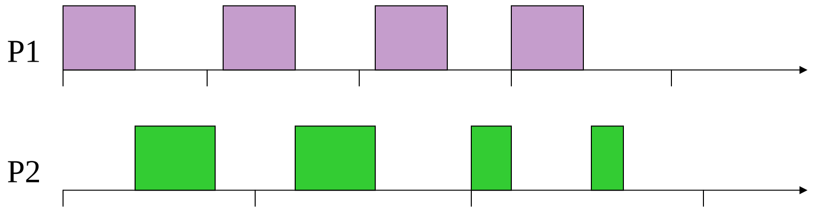


No process is ready…

EDF example



EDF example



EDF analysis

- EDF can use 100% of CPU for worst case
- But EDF may miss deadlines.

EDF implementation

- On each timer interrupt:
  - compute time to deadline;
  - choose process closest to deadline.
- Generally considered too expensive to use in practice, unless the task count is small

Priority Inversion

- Priority Inversion: low-priority process keeps high-priority process from running.
- Improper use of system resources can cause scheduling problems:
  - Low-priority process grabs I/O device.
  - High-priority device needs I/O device, but can’t get it until low-priority process is done.
- Can cause deadlock.

Solving priority inversion

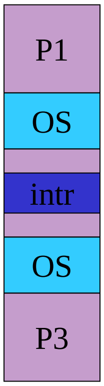
- Give priorities to system resources.
- Have process inherit the priority of a resource that it requests.
  - Low-priority process inherits priority of device if higher.

Context-switching time

- Non-zero context switch time can push limits of a tight schedule.
- Hard to calculate effects---depends on order of context switches.
- In practice, OS context switch overhead is small.

What about interrupts?

- Interrupts take time away from processes.
- Other event processing may be masked during interrupt service routine (ISR)
- Perform minimum work possible in the interrupt handler.



Device processing structure

- Interrupt service routine (ISR) performs minimal I/O.
  - Get register values, put register values.
- Interrupt service process/thread performs most of device function.

Evaluating performance

- May want to test
  - context switch time assumptions on real platform
  - scheduling policy

Processes and caches

- Processes can cause additional caching problems.
  - Even if individual processes are well-behaved, processes may interfere with each other.
- Worst-case execution time with **bad cache behavior** is usually much worse than execution time with good cache behavior.

Fixing scheduling problems

- What if your set of processes is unschedulable?
  - Change deadlines in requirements.
  - Reduce execution times of processes.
  - Get a faster CPU
  - **Get an Accelerator**

Outline

- Introduction
- When to Use Accelerators
  - Real Time Scheduling
- ➡ Accelerated System Design
  - Architecture Selection
  - Partitioning and Scheduling
- Key Recent Trends

Accelerated system design

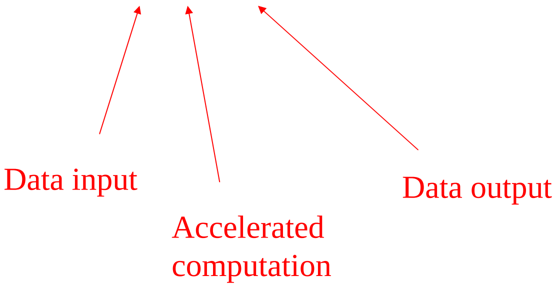
- First, determine that the system really needs to be accelerated.
  - How much faster is the accelerator on the core function?
  - How much data transfer overhead?
- Design the accelerator itself.
- Design CPU interface to accelerator.

Performance analysis

- Critical parameter is speedup: how much faster is the system with the accelerator?
- Must take into account:
  - Accelerator execution time.
  - Data transfer time.
  - Synchronization with the master CPU.

Accelerator execution time

- Total accelerator execution time:
  - $t_{accel} = t_{in} + t_x + t_{out}$



Data input

Accelerated computation

Data output

Data input/output times

- Bus transactions include:
  - flushing register/cache values to main memory;
  - time required for CPU to set up transaction;
  - overhead of data transfers by bus packets, handshaking, etc.

Accelerator speedup

- Assume loop is executed *n* times.
- Compare accelerated system to non-accelerated system:
  - Saved Time = $n(t_{CPU} - t_{accel})$
  - = $n[t_{CPU} - (t_{in} + t_x + t_{out})]$

Execution time of equivalent function on CPU

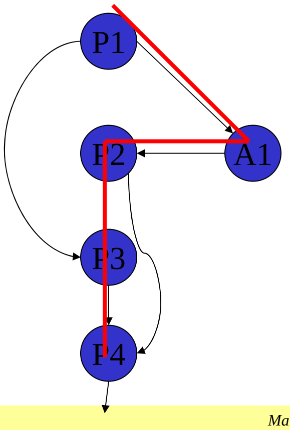
- Speed-Up = Original Ex. Time / Accelerated Ex.Time
- Speed-Up = t_{CPU} / t_{accel}

Single- vs. multi-threaded

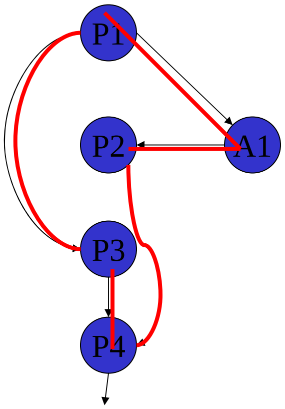
- One critical factor is available parallelism:
  - single-threaded/blocking: CPU waits for accelerator;
  - multithreaded/non-blocking: CPU continues to execute along with accelerator.
- To multithread, CPU must have useful work to do.
  - But software must also support multithreading.

Total execution time

- Single-threaded:

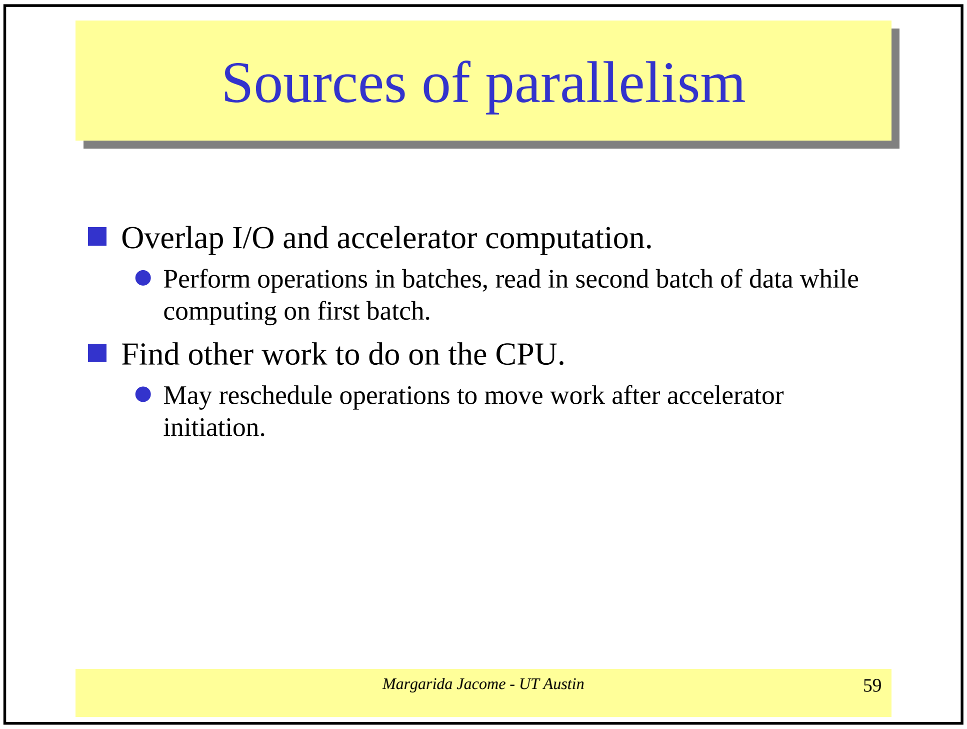


- Multi-threaded:



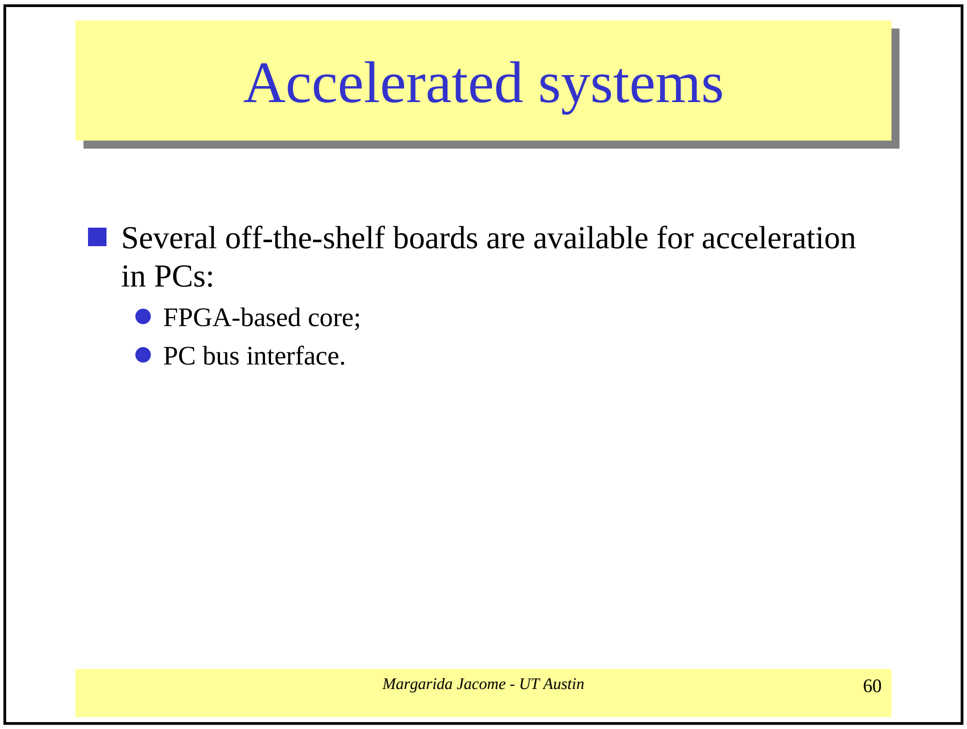
Execution time analysis

- Single-threaded:
  - Count execution time of all component processes.
- Multi-threaded:
  - Find longest path through execution.



Sources of parallelism

- Overlap I/O and accelerator computation.
  - Perform operations in batches, read in second batch of data while computing on first batch.
- Find other work to do on the CPU.
  - May reschedule operations to move work after accelerator initiation.



Accelerated systems

- Several off-the-shelf boards are available for acceleration in PCs:
  - FPGA-based core;
  - PC bus interface.

Accelerator/CPU interface

- Accelerator registers provide control registers for CPU.
- Data registers can be used for small data objects.
- Accelerator may include special-purpose read/write logic (DMA hardware)
  - Especially valuable for large data transfers.

Caching problems

- Main memory provides the primary data transfer mechanism to the accelerator.
- Programs must ensure that caching does not invalidate main memory data.
  - CPU reads location S.
  - Accelerator writes location S.
  - CPU writes location S.

BAD

(program will not see the value of S stored in the cache)

The bus interface may provide mechanisms for accelerators to tell the CPU of required cache changes…

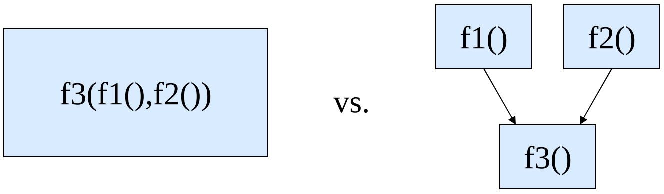
Synchronization

- As with cache, main memory writes to shared memory may cause invalidation:
  - CPU reads S.
  - Accelerator writes S.
  - CPU write S.

Many CPU buses implement test-and-set atomic operations that the accelerator can use to implement a semaphore. This can serve as a highly efficient means of synchronizing inter-process Communications (IPC).

Partitioning/Decomposition

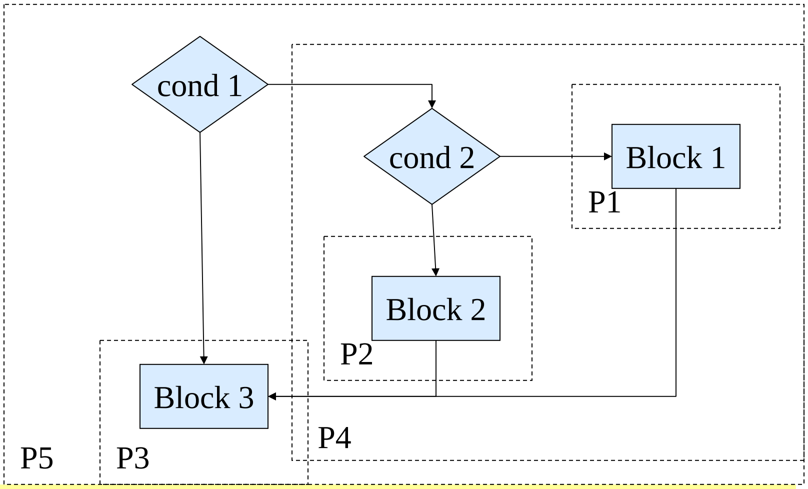
- Divide functional specification into units.
  - Map units onto PEs.
  - Units may become processes.
- Determine proper level of parallelism:

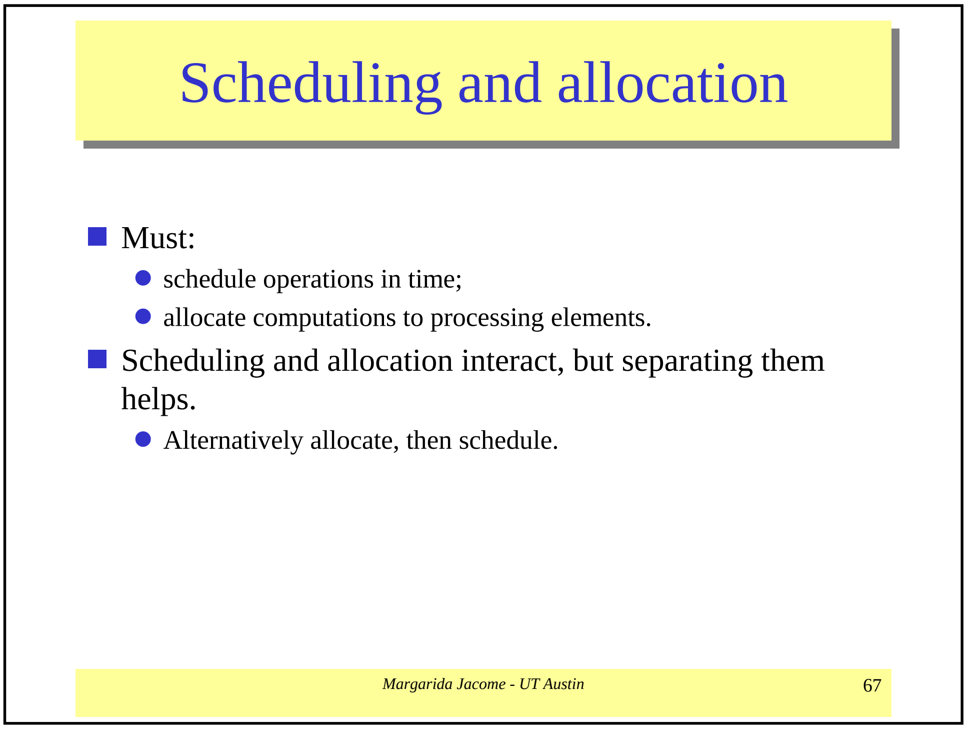


“Typical” Decomposition Methodology

- Divide Control-Data Flow Graph (CDFG) into pieces, shuffle functions between pieces.
- Hierarchically decompose CDFG to identify possible partitions.

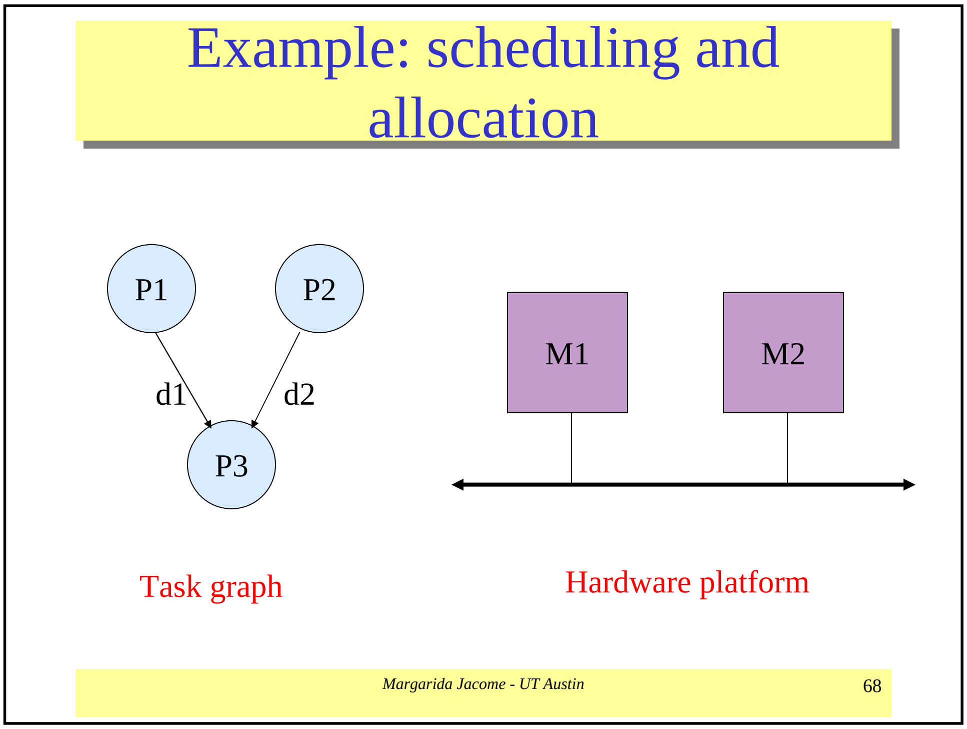
Decomposition example





Scheduling and allocation

- Must:
  - schedule operations in time;
  - allocate computations to processing elements.
- Scheduling and allocation interact, but separating them helps.
  - Alternatively allocate, then schedule.



Example: scheduling and allocation

P1

P2

d1

d2

P3

M1

M2

Task graph

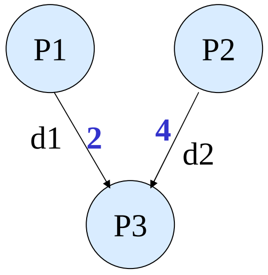
Hardware platform

Example process execution times

	M1	M2
P1	5	5
P2	5	6
P3	--	5

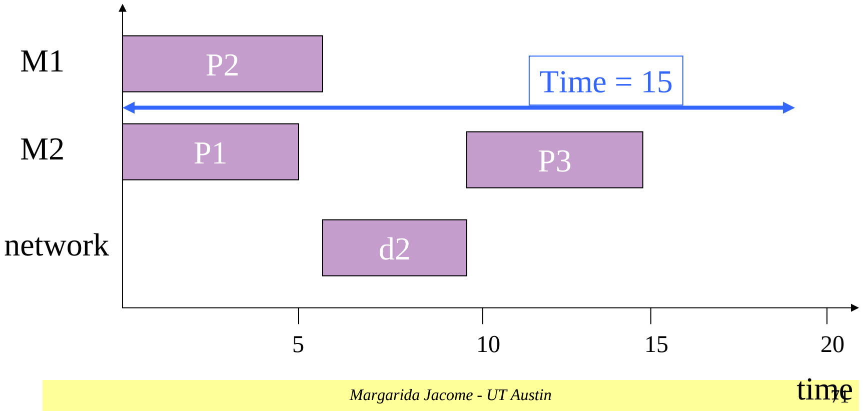
Example communication model

- Assume communication within PE is free.
- Cost of communication from P1 to P3 is d1 =2; cost of P2 to P3 communication is d2 = 4.



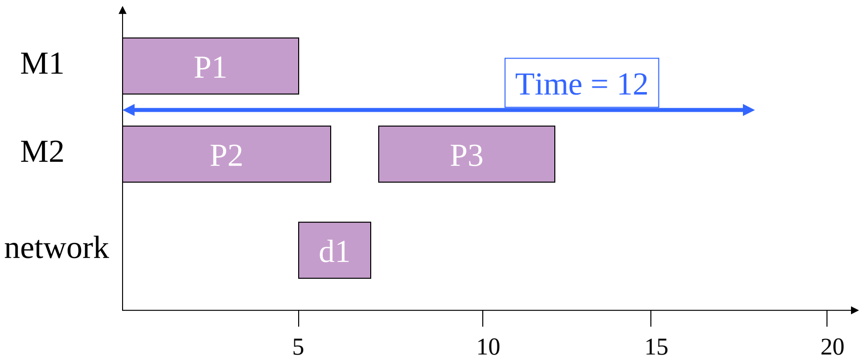
First design

- Allocate P2 -> M1; P1, P3 -> M2.



Second design

- Allocate P1 -> M1; P2, P3 -> M2:



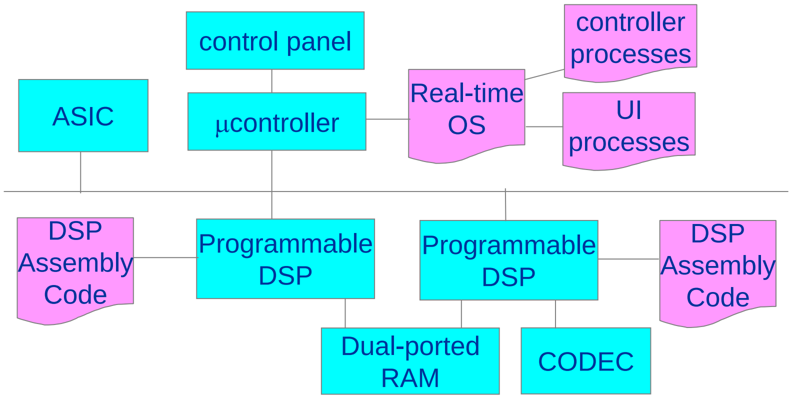
System integration and debugging

- Try to debug the CPU/accelerator interface separately from the accelerator core.
- Build scaffolding to test the accelerator (Hardware Abstraction Layer is a good place for this functionality, under compile switches)
- Hardware/software co-simulation can be useful.

Outline

- Introduction
- When to Use Accelerators
  - Real Time Scheduling
- Accelerated System Design
  - Architecture Selection
  - Partitioning and Scheduling
- Key Recent Trends

Complexity and Heterogeneity



control panel

controller processes

Real-time OS

ASIC

µcontroller

UI processes

DSP Assembly Code

Programmable DSP

Programmable DSP

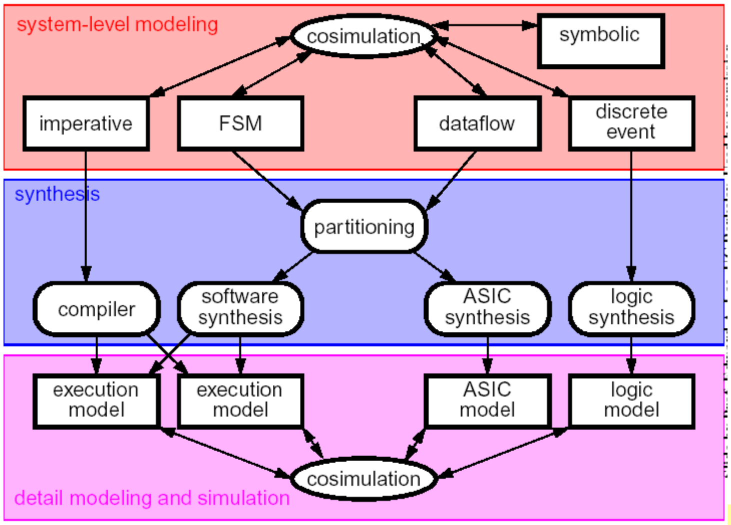
DSP Assembly Code

Dual-ported RAM

CODEC

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

Handling Heterogeneity



system-level modeling

cosimulation

symbolic

imperative

FSM

dataflow

discrete event

synthesis

partitioning

compiler

software synthesis

ASIC synthesis

logic synthesis

execution model

execution model

ASIC model

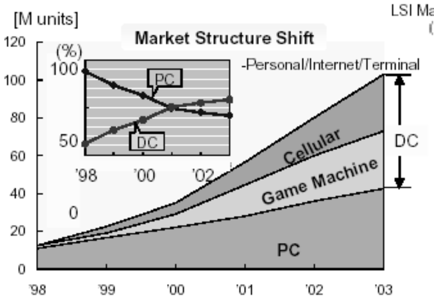
logic model

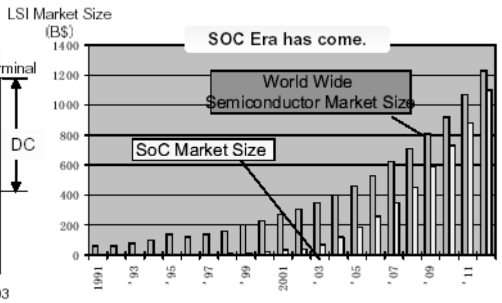
cosimulation

detail modeling and simulation

From Lee (Berkeley)

Industrial Structure Shift (from Sony)





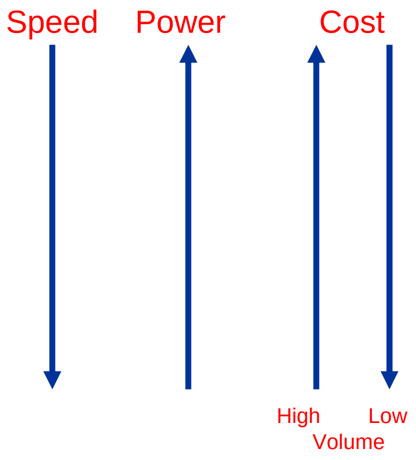
- Current Percentage of SoC Ratio is under 10%.
  ⇒40% in 2005, 70‒80% in 2010
- SoC is "single-seat constituency ", "take or not".
- Key Factor is the Synergy between Semiconductor & Set Divisions.

90's
- PC

00's
- High Performance ‒ Game Machine
- Low Power ‒ Cellular

Many Implementation Choices

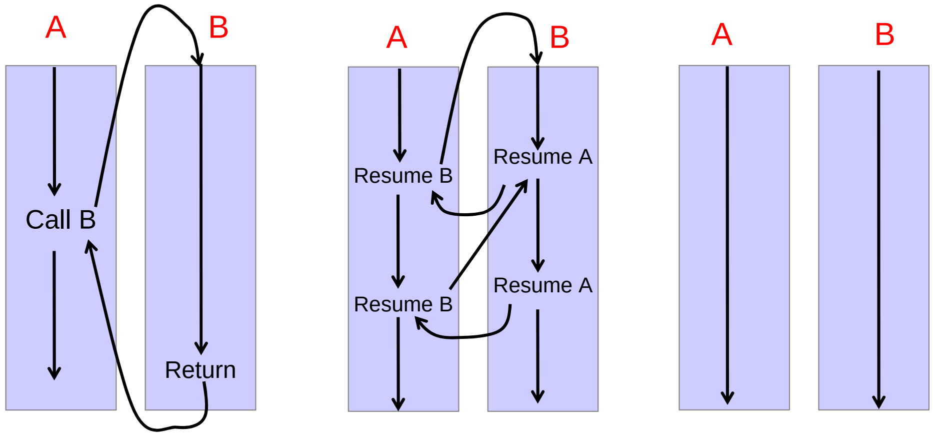
- Microprocessors
- Domain-specific processors
  - DSP
  - Network processors
  - Microcontrollers
- ASIPs
- Reconfigurable SoC
- FPGA
- Gatearray
- ASIC



Hardware vs. Software Modules

- Hardware = functionality implemented via a custom architecture (e.g. datapath + FSM)
- Software = functionality implemented in software on a programmable processor
- Key differences:
  - Multiplexing
    - software modules multiplexed with others on a processor
      - e.g. using an OS
    - hardware modules are typically mapped individually on dedicated hardware
  - Concurrency
    - processors usually have one "thread of control"
    - dedicated hardware often has concurrent datapaths

Multiplexing Software Modules



SUBROUTINES
Hierarchical
Sequential

COROUTINES
Symmetric
Sequential

PROCESSES
Symmetric
Concurrent

Modularity
Complexity

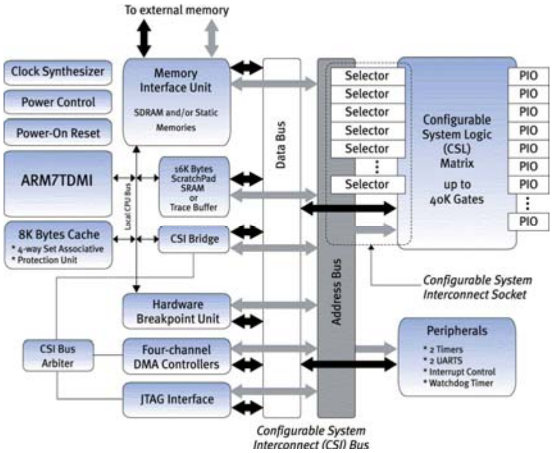
Many Types of Programmable Processors

- Past/Now
  - ◆ Microprocessor
  - ◆ Microcontroller
  - ◆ DSP
  - ◆ Graphics Processor
- Now / Future
  - ◆ Network Processor
  - ◆ Sensor Processor
  - ◆ Cryptoprocessor
  - ◆ Game Processor
  - ◆ Wearable Processor
  - ◆ Mobile Processor

Application-Specific Instruction Processors (ASIPs)

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

Reconfigurable SoC



Triscend's A7 CSoC

Other Examples

Atmel's FPSLIC (AVR + FPGA)

Altera's Nios (configurable RISC on a PLD)

H/W-S/W Architecture

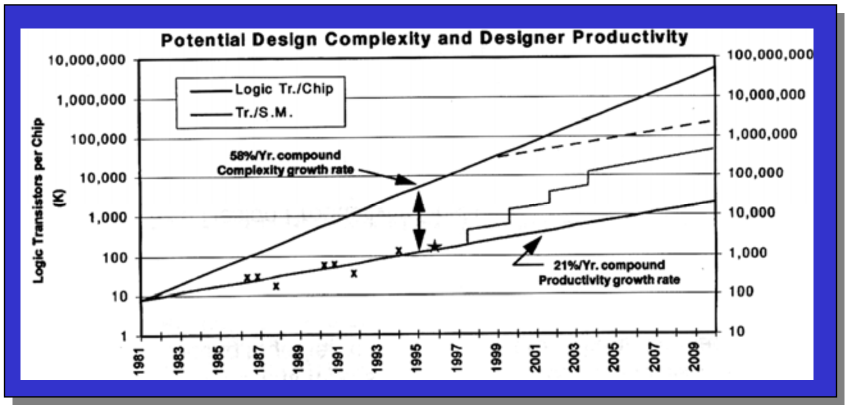
- A significant part of the problem is deciding which parts should be in s/w on programmable processors, and which in specialized h/w
- Today:
  - Ad hoc approaches based on earlier experience with similar products, & on manual design
  - H/W-S/W partitioning decided at the beginning, and then designs proceed separately

Embedded System Design

- CAD tools take care of HW fairly well (at least in relative terms)
  - Although a productivity gap emerging
- But, SW is a different story…
  - HLLs such as C help, but can't cope with complexity and performance constraints

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

Productivity Gap in Hardware Design

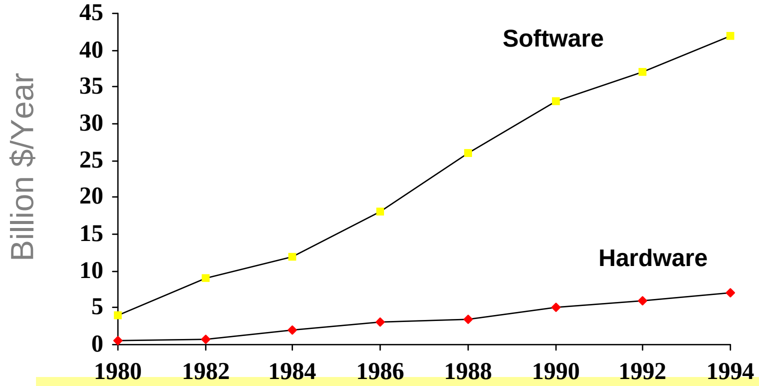


Source: sematech97

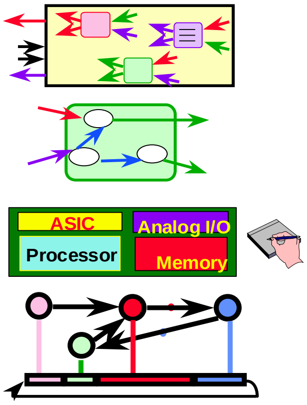
A growing gap between design complexity and design productivity

Situation Worse in S/W

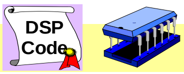
DoD Embedded System Costs



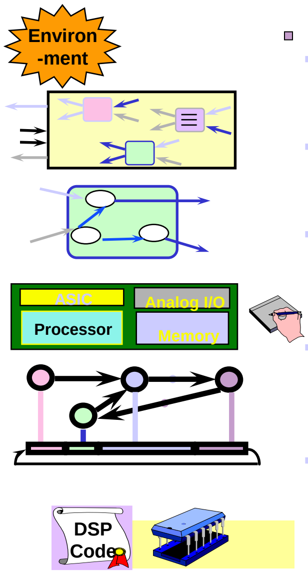
Embedded System Design from a Design Technology Perspective



- Intertwined subtasks
  - Specification/modeling
  - H/W & S/W partitioning
  - Scheduling & resource allocations
  - H/W & S/W implementation
  - Verification & debugging
- Crucial is the co-design and joint optimization of hardware and software

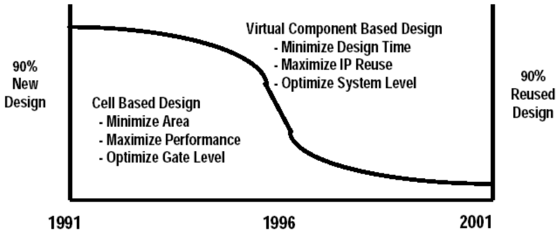


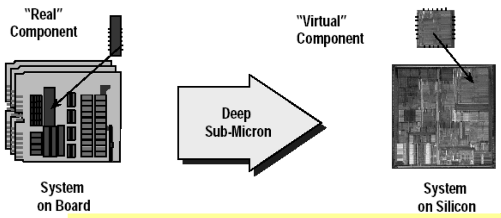
Embedded System Design Flow



- **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 suitable programmable processor.
- **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.

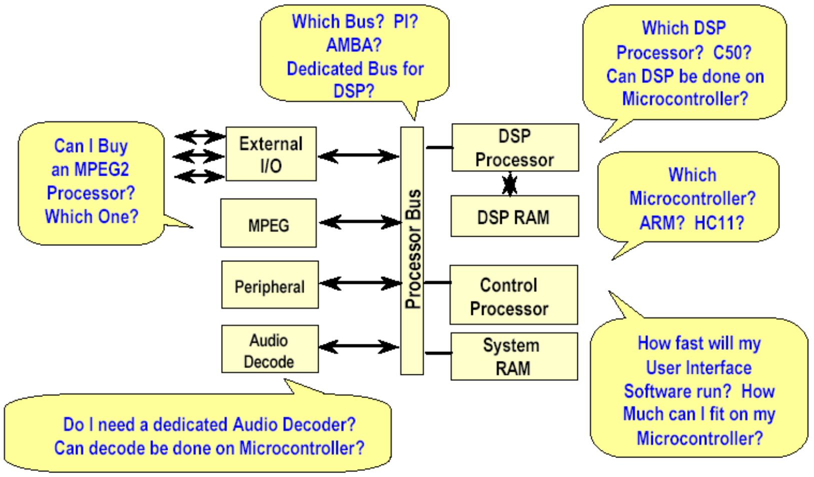
On-going Paradigm Shift in Embedded System Design





- Change in business model due to SoCs
  - Currently many IC companies have a chance to sell devices for a single board
  - In future, a single vendor will create a System-on-Chip
  - But, how will it have knowledge of all the domains?
- Component-based design
  - Components encapsulate the intellectual property
- Platforms
  - Integrated HW/SW/IP
  - Application focus
  - Rapid low-cost customization

IP-based Design



Which Bus? PI? AMBA? Dedicated Bus for DSP?

Which DSP Processor? C50? Can DSP be done on Microcontroller?

Can I Buy an MPEG2 Processor? Which One?

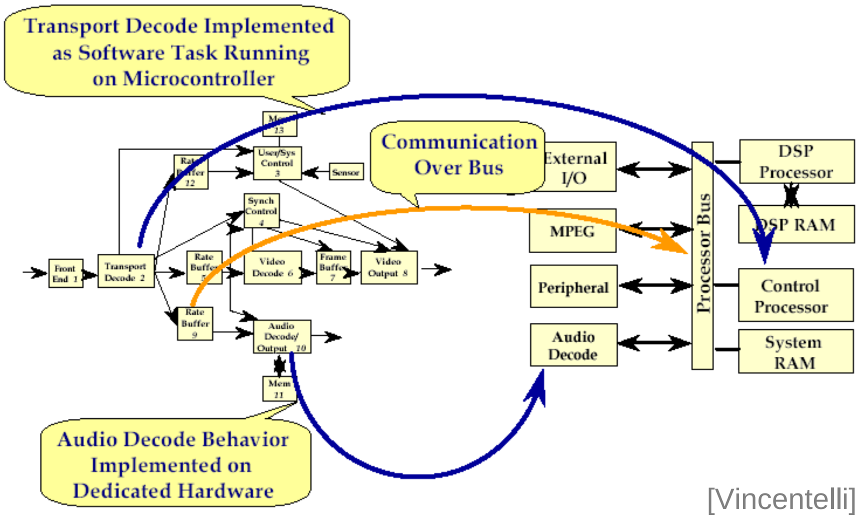
Which Microcontroller? ARM? HC11?

How fast will my User Interface Software run? How Much can I fit on my Microcontroller?

Do I need a dedicated Audio Decoder? Can decode be done on Microcontroller?

[Vincentelli]

Map from Behavior to Architecture



Transport Decode Implemented as Software Task Running on Microcontroller

Communication Over Bus

Audio Decode Behavior Implemented on Dedicated Hardware

[Vincentelli]