

EE382M.20: System-on-Chip (SoC) Design

Lecture 6 – Task Partitioning

Sources:

Prof. Margarida Jacome, UT Austin

Prof. Lothar Thiele, ETH Zürich

Andreas Gerstlauer

Electrical and Computer Engineering

University of Texas at Austin

gerstl@ece.utexas.edu



The University of Texas at Austin

Electrical and Computer Engineering

Cockrell School of Engineering

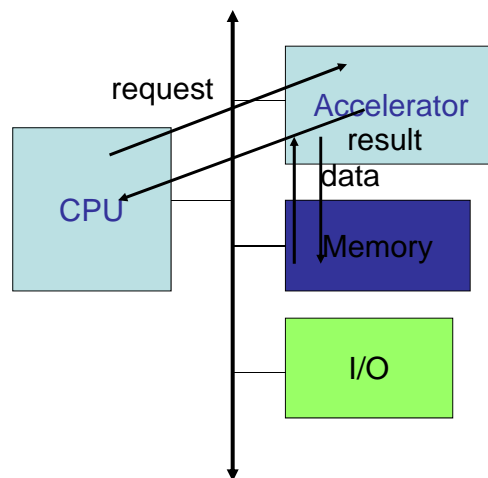
Lecture 6: Outline

- **Accelerated system design**
 - When to use accelerators
 - Performance analysis
- **Partitioning**
 - Decomposition
 - Constructive heuristics
 - Iterative heuristics
- **System-level design**
 - MPSoC trends

Hardware vs. Software Modules

- **Hardware**
 - Functionality implemented via a custom architecture (e.g. datapath + FSM)
 - **Software**
 - Functionality implemented on a programmable processor (datapath + programmable control)
- **Key differences**
- **Concurrency**
 - Processors usually have one “thread of control”
 - Dedicated hardware often has concurrent datapaths
 - **Multiplexing**
 - Software modules multiplexed with others on a processor (e.g. OS)
 - Hardware modules are typically mapped individually on dedicated hardware blocks

Accelerated System Architecture

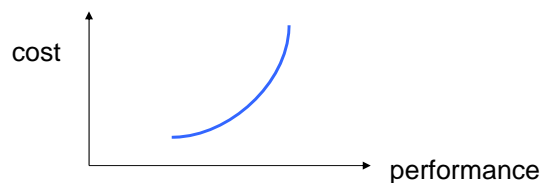


Accelerators

- **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 (ASIC)
 - Field-programmable gate array (FPGA).
 - Standard component.
 - Example: graphics processor.
 - SoCs enable multiple accelerators, peripherals, and some memory to be placed with a CPU on a single chip

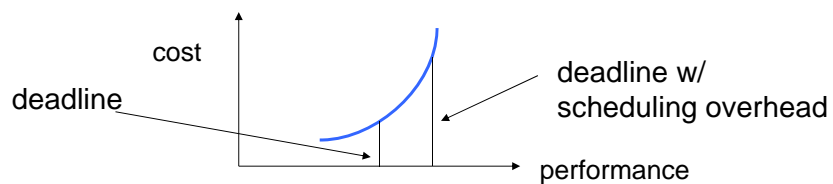
Why Accelerators?

- **Better cost/performance**
 - Custom logic may be able to perform operation faster or at lower power than a CPU of equivalent cost
 - Better at real-time, I/O, streaming, parallelism
 - CPU cost is a non-linear function of performance
 - May not be able to do the work on even the largest CPU



Why Accelerators? (cont'd)

- **Better real-time performance**
 - Put time-critical functions on less-loaded processing elements
 - Scheduling utilization is '*limited*'---extra CPU cycles must be reserved to meet deadlines. (see *previous lecture*)



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
- **Total accelerator execution time**

$$t_{\text{accel}} = t_{\text{in}} + t_x + t_{\text{out}}$$

Data input
Accelerated computation
Data output

Accelerator Speedup

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

Execution time of equivalent function on CPU
 - Speed-Up = Original Ex. Time / Accelerated Ex. Time
 - Speed-Up = $t_{\text{CPU}} / t_{\text{accel}}$
- Data input/output times include
 - flushing register/cache values to main memory;
 - time required for CPU to set up transaction;
 - data transfer overhead for bus packets, handshaking, etc.

Accelerator/CPU Interface

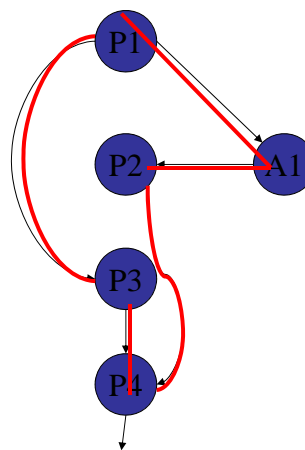
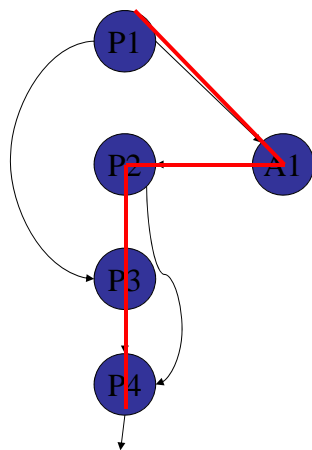
- Data transfers
 - Accelerator registers provide control registers for CPU
 - Shared memory region for data exchange
 - Data registers can be used for small data objects
 - Accelerator may include special-purpose read/write logic (bus mastering DMA hardware)
 - Especially valuable for large data transfers
- Caching problems
 - CPU might not see memory writes by the accelerator
 - Invalidate cache lines or disable caching of shared regions
- Synchronization
 - Concurrent accesses to shared variables
 - Semaphores using atomic test & set bus operations

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
- **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.

Execution Time Analysis

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

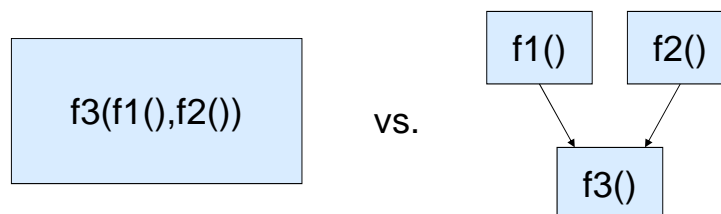


Lecture 6: Outline

- ✓ Accelerated system design
 - ✓ When to use accelerators
 - ✓ Performance analysis
- HW/SW partitioning
 - Decomposition
 - Constructive heuristics
 - Iterative heuristics
- System-level design
 - MPSoC trends

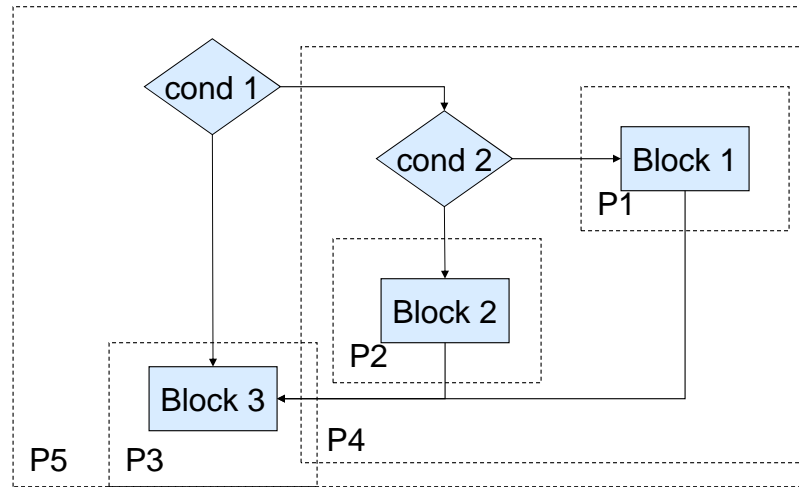
Decomposition

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



Decomposition Example

- Divide program into Control-Data Flow Graph (CDFG)
- Hierarchically decompose CDFG to identify partitions



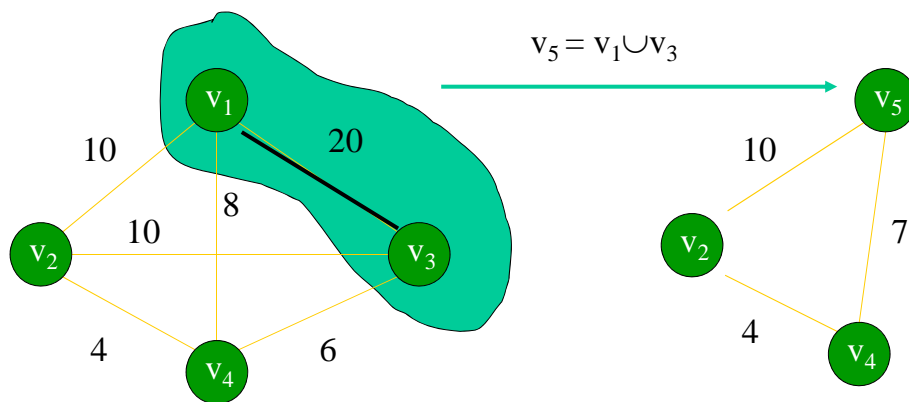
Partitioning

- Assign tasks (objects) to processors (partitions) such that all objects are assigned to unique partitions
 - Minimize communication cost (graph partitioning)
 - Minimize partition count (bin packing)
 - Partition size, partition count, etc. constraints
- **Exact methods**
 - Exhaustive enumeration, integer linear programming (ILP)
- **Constructive heuristics**
 - Random mapping, hierarchical clustering
- **Iterative heuristics**
 - Hill climbing, Kernighan-Lin, simulated annealing

Constructive Methods

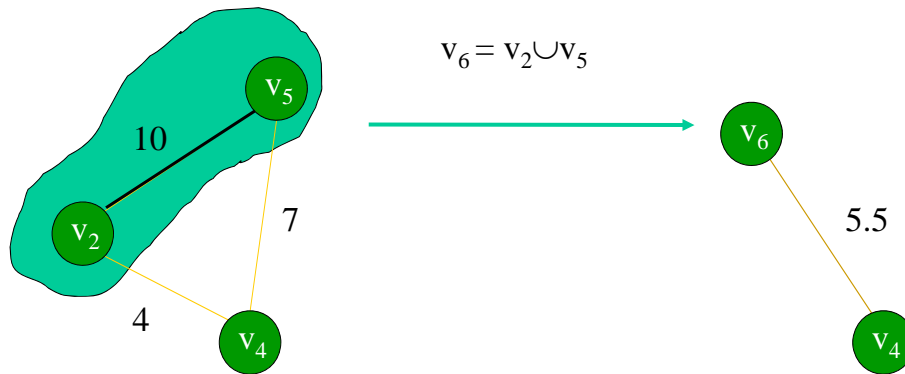
- **Construct solution one by one**
 - Visit every object once
 - Can generate a starting partition for iterative methods
 - Shows the difficulty of finding proper closeness functions
- **Random mapping**
 - Each object is assigned to a block randomly
- **Hierarchical clustering**
 - Stepwise grouping of objects
 - Closeness function determines how desirable it is to group two objects

Hierarchical Clustering - Example (1)



Closeness function: arithmetic mean of weights

Hierarchical Clustering - Example (2)

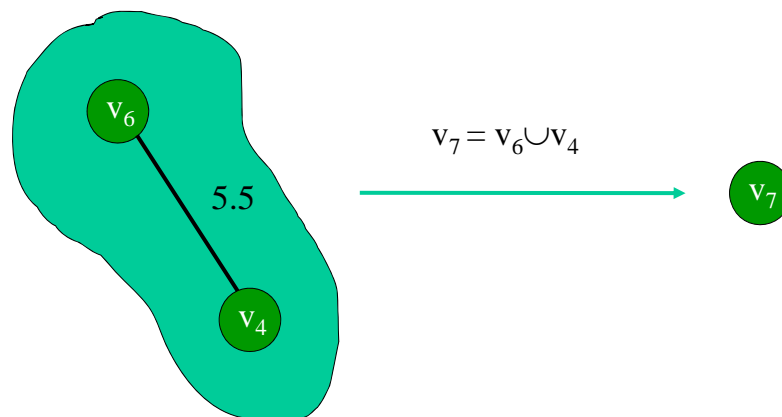


EE382M.20: SoC Design, Lecture 6

© Lothar Thiele, ETH Zürich

19

Hierarchical Clustering - Example (3)

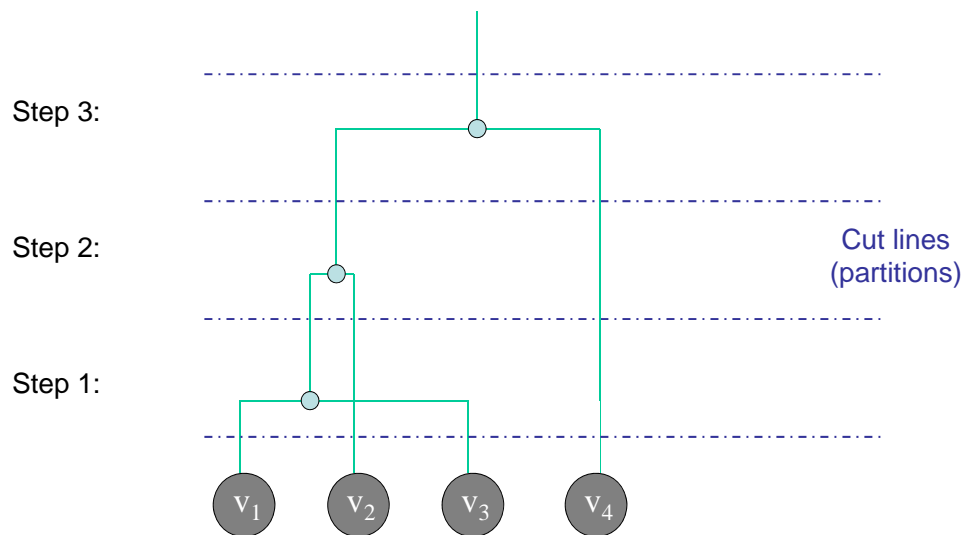


EE382M.20: SoC Design, Lecture 6

© Lothar Thiele, ETH Zürich

20

Hierarchical Clustering - Example (4)



EE382M.20: SoC Design, Lecture 6

© Lothar Thiele, ETH Zürich

21

Iterative Methods

- **Principles**
 - Start with some initial solution
 - Search neighborhood (similar solutions), select candidate and make local change based on fitness/cost function
 - End on stopping criterion
- **Simple iterative improvement or “hill climbing”**
 - Select candidate with best improvement in cost
 - Stop when no candidate with lower cost is found
- **Kernighan-Lin**
 - More exhaustive search to escape local optima

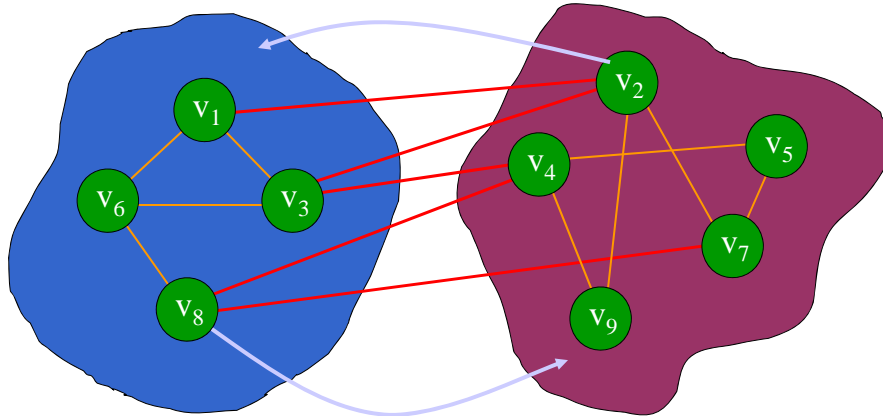
EE382M.20: SoC Design, Lecture 6

© Lothar Thiele, ETH Zürich

22

Iterative Improvement (Hill Climbing)

- **Simple greedy heuristic**
 - Until there is no improvement in cost: re-group a pair of objects which leads to the largest gain in cost



Example: Cost = number of edges crossing the partitions
Before re-group: 5 ; after re-group: 4 ; gain = 1

Kernighan-Lin

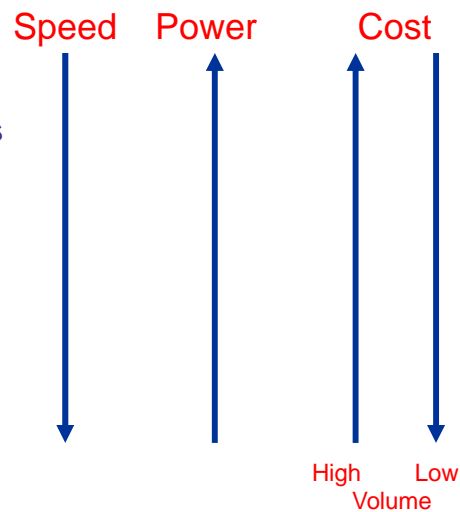
- **Problem**
 - Simple greedy heuristic can get stuck in a local minimum
- **Kernighan-Lin algorithm**
 - As long as a better partition is found
 - From all possible pairs of objects, virtually re-group the “best” (lowest cost of the resulting partition)
 - From the remaining not yet touched objects, virtually re-group the “best” pair, etc.,
 - Continue until all objects have been re-grouped
 - From these $n/2$ partitions take the one with smallest cost and actually perform the corresponding re-group operations. $O(n^2 \log n)$ complexity
 - Still can get stuck in local minimum
 - Among sequences of moves
- **More complex strategies**
 - Randomize search, e.g. simulated annealing

Lecture 6: Outline

- ✓ **Accelerated system design**
 - ✓ When to use accelerators
 - ✓ Performance analysis
- ✓ **HW/SW partitioning**
 - ✓ Decomposition
 - ✓ Partitioning heuristics
- **System-level design**
 - MPSoC trends

Many More Implementation Choices

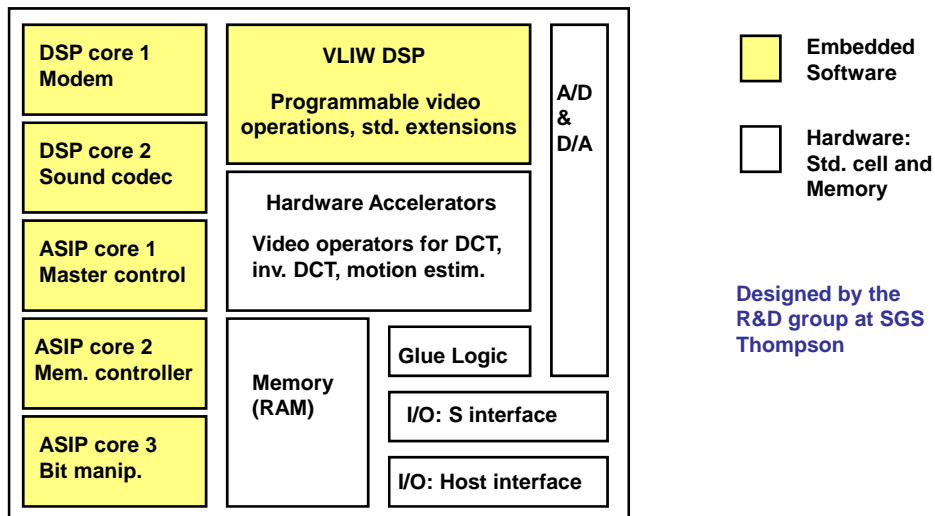
- **Microprocessors**
 - Microcontrollers
- **Domain-specific processors**
 - DSP
 - Graphics/network processors
- **ASIPs**
- **Reconfigurable SoC**
- **FPGA**
- **Gatearray**
- **ASIC**



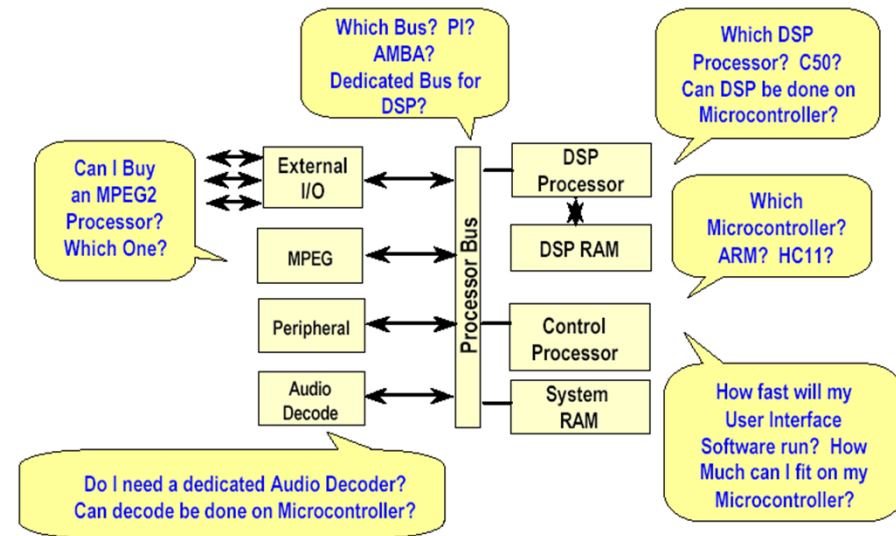
Heterogeneous Processors

- **Many types of programmable processors**
 - Past/now: micro-processor/-controller, DSP
 - Now/future: graphics, network, crypto, game, ... processor
- **Application-specific instruction-set processor (ASIP)**
 - 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

MPSoC: Video Telephone

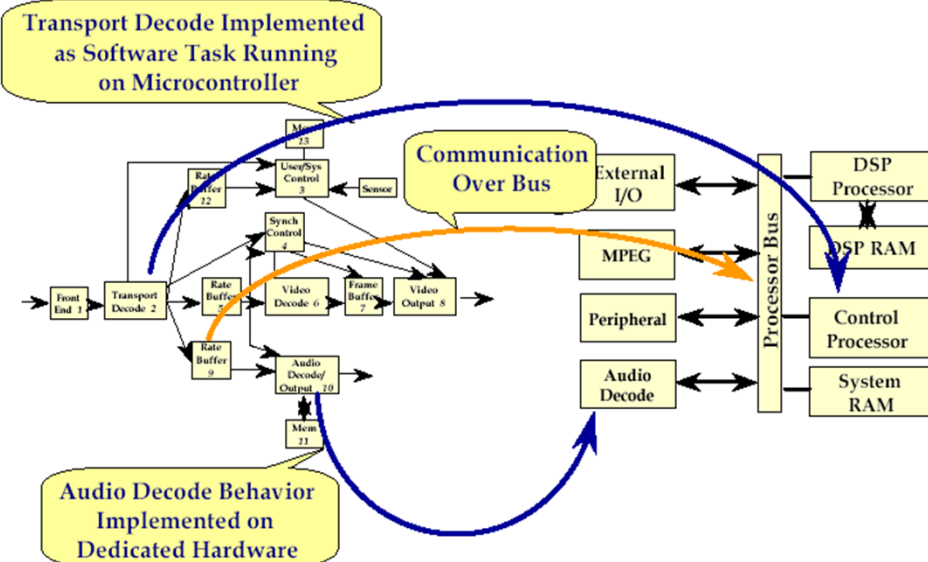


IP-Based Design



Source: A. Sangiovanni-Vincentelli, UC Berkeley

Platform Mapping



Source: A. Sangiovanni-Vincentelli, UC Berkeley

MPSoC Synthesis Tasks

- **Recap: Mapping**
 - Allocate resources (hardware/software processors)
 - Bind computations to resources
 - Schedule operations in time
 - Partitioning = (allocation +) binding
 - Mapping = binding + scheduling
- **Allocation, scheduling and binding interact**
 - Even more so in MPSoC case