

ECE382M.20: System-on-Chip (SoC) Design

Lecture 3 – HW/SW Co-Design

Sources:

Prof. Margarida Jacome, UT Austin

Andreas Gerstlauer

Electrical and Computer Engineering

The University of Texas at Austin

gerstl@ece.utexas.edu



The University of Texas at Austin
Chandra Department of Electrical
and Computer Engineering
Cockrell School of Engineering

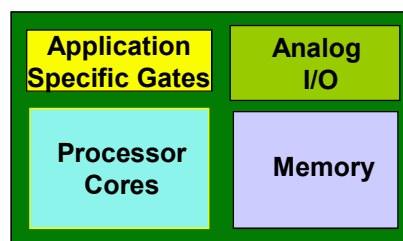
Lecture 3: Outline

- **Introduction**
 - SoC architecture
- **Accelerated system design**
 - When and for what to use accelerators
 - Performance analysis
- **HW/SW co-design**
 - Specification
 - Analysis
 - Synthesis

SoC Architecture

- **Employ a combination of**

- SW on programmable processors
 - Flexibility, complexity
- Application-specific, custom HW
 - Performance, low power
- Memory
 - On-chip SRAM or eDRAM
- I/O
 - Transducers, sensors, actuators, A/D & D/A converters
 - Interact with analog, continuous-time environment



- **Heterogeneous, accelerator-rich system architecture**

- CPUs, DSPs, GPUs, micro-controllers
- ASICs & field programmable gate arrays (FPGAs)

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

Hardware Accelerator Taxonomy

- **Accelerator vs. co-processor**
 - Tightly-coupled, fine-grain co-processors
 - Executes instructions dispatched by the CPU, integrated into pipeline
 - Loosely-coupled, coarse-grain accelerators
 - Executes thread as separate device on bus, controlled via registers
- **Accelerator implementations**
 - Application-specific integrated circuit (ASIC)
 - Field-programmable gate array (FPGA)
 - Standard component
 - Example: graphics processor unit (GPU)
- **SoCs enable multiple accelerators, peripherals, and some memory to be placed with a CPU on a single chip**

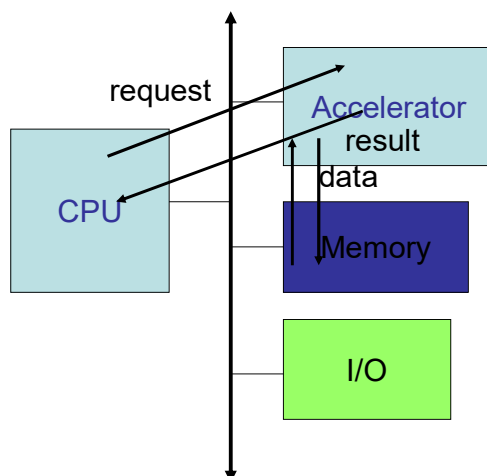
ECE382M.20: SoC Design, Lecture 3

© Margarida Jacome, UT Austin

5

Accelerated System Architecture

- **Most basic SoC architecture**
 - CPU + accelerator (+ memory + I/O)



ECE382M.20: SoC Design, Lecture 3

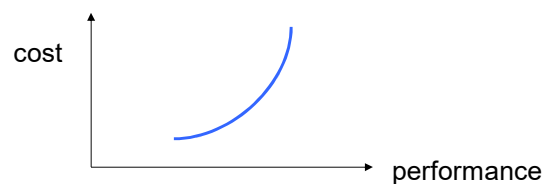
© Margarida Jacome, UT Austin

6

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



ECE382M.20: SoC Design, Lecture 3

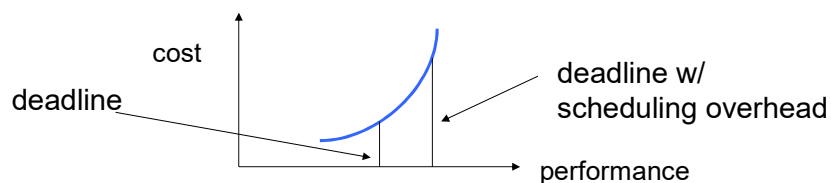
© Margarida Jacome, UT Austin

7

Why Accelerators? (cont'd)

- **Better real-time performance**

- Time-critical functions on less-loaded processing elements
- Dynamic CPU effects make it hard to predict timing
 - Scheduling overhead and engineering margin



ECE382M.20: SoC Design, Lecture 3

© Margarida Jacome, UT Austin

8

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

ECE382M.20: SoC Design, Lecture 3

© Margarida Jacome, UT Austin

9

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.

ECE382M.20: SoC Design, Lecture 3

© Margarida Jacome, UT Austin

10

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

ECE382M.20: SoC Design, Lecture 3

© Margarida Jacome, UT Austin

11

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.

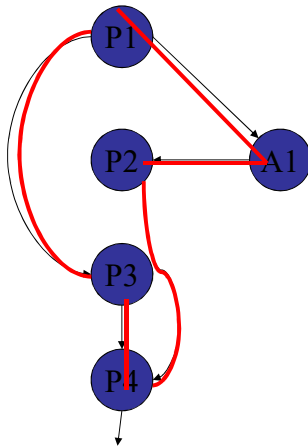
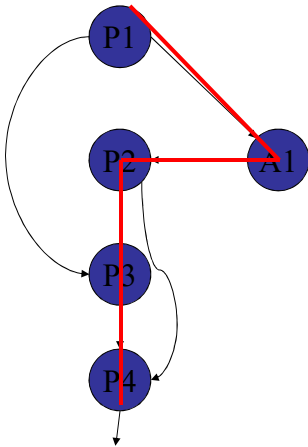
ECE382M.20: SoC Design, Lecture 3

© Margarida Jacome, UT Austin

12

Execution Time Analysis

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



Lecture 3: Outline

- ✓ Introduction
 - ✓ SoC design
- ✓ Accelerated system design
 - ✓ When and for what to use accelerators
 - ✓ Performance analysis
- **HW/SW co-design**
 - Specification
 - Analysis
 - Synthesis

Embedded SoC Design

- The design of an embedded SoC consists of correctly implementing a specific set of *functions* while satisfying constraints on
 - Performance
 - Dollar cost
 - Energy consumption, power dissipation
 - Weight, etc.

The choice of a *system architecture* impacts whether designers will implement a *function* as custom hardware or as (embedded) software running on a programmable component (processor).

Design Problem

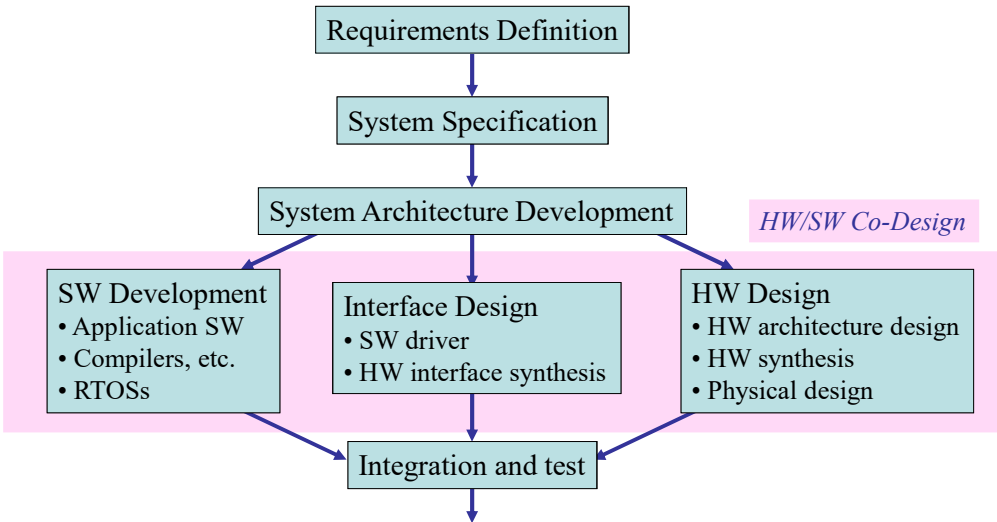
- Design a heterogeneous multiprocessor architecture that satisfies the design requirements
 - Use computational unit(s) dedicated to some functions
 - Processing elements (PE): CPU, custom hardware accelerators
 - Program the system
- A significant part of the design problem is deciding which parts should be in SW on programmable processors, and which in specialized HW
 - Deciding the HW/SW architecture
- Ad-hoc approaches today
 - Based on earlier experience with similar products
 - HW/SW partitioning decided a priori, designed separately

Design Automation

- **Computer-Aided Design (CAD)**
Electronic Design Automation (EDA)
 - Tools take care of HW fairly well (at least in relative terms)
 - Productivity gap emerging
- **Situation in SW is worse**
 - HLLs such as C help, but can't cope with exponential increase in complexity and performance constraints

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

System-Level Design Process



HW/SW Co-Design

- **Use additional computational unit(s) dedicated to some functions**
 - Hardwired logic, extra CPU
- **Automated design & optimization of HW/SW systems**
 - Specification
 - Modeling
 - Performance analysis
 - Synthesis
 - HW/SW partitioning (resource allocation & binding)
 - Scheduling
 - HW & SW implementation
 - SW compilation
 - HW synthesis
 - Validation
 - Integration, verification & debugging

Lecture 5: Outline

- ✓ **Introduction**
 - ✓ Embedded SoC design
- **HW/SW co-design**
 - Specification
 - Analysis
 - Synthesis

System Specification

- **Describe the desired behavior of a design as a relation between a set of inputs and a set of outputs**
 - This relation may be informal, even expressed in natural language
 - Such informal, ambiguous specifications may result in unnecessary redesigns...
- **Formal Models of Computation (MoCs)**
 - Computation/behavior
 - Communication
 - Concurrency
 - Time/order
 - Heterogeneity, composability
 - Implementability

Main MoCs for Embedded Systems

- **Programming models**
 - Imperative & declarative
 - Synchronous/reactive
- **Process-based models**
 - Discrete event
 - Kahn Process Networks (KPNs)
 - (Synchronous) Dataflow models ((S)DF)
- **State-based models**
 - Finite State Machines (FSM)
 - Hierarchical, Concurrent State Machines (HCFSM)
 - Petri Nets
- **ECE382N.23: Embedded System Design & Modeling**

Task Graph Model

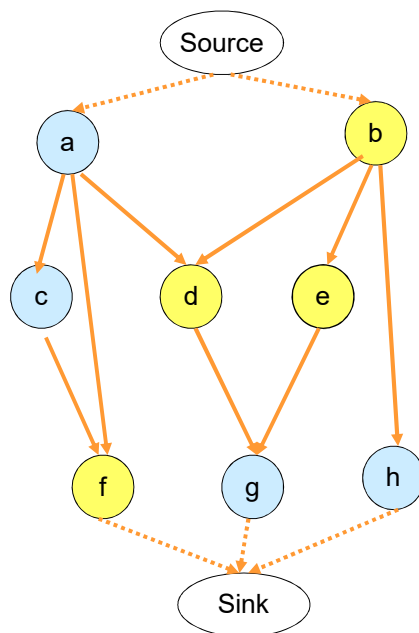
- **A graph representation of the application specification**
 - Derived from data dependency based representation commonly utilized in compilers
- **Application is specified by a graph $G(V,E)$**
 - V is the set of tasks
 - $t(v,r)$ gives the run-time of “ v ” on a processing element “ r ”
 - E is the set of directed edges
 - $e(u,v)$ implies data produced by u is consumed by v
 - v cannot begin execution before u has finished execution
 - Execution constraints
 - Deadlines, rates, latencies
- **Data-dominated application model**
 - Multimedia and network processing applications can be specified by this model

ECE382M.20: SoC Design, Lecture 3

© 2026 A. Gerstlauer

23

Task Graph Example



- **Assign weights to nodes and edges**
 - Cost, delays
- **Constraints for nodes or whole graph**
 - Source-to-sink delay
- **Analysis & synthesis**
 - Partitioning
 - Real-Time Scheduling
- **Amdahl's law**

ECE382M.20: SoC Design, Lecture 3

© 2026 A. Gerstlauer

24

Lecture 3: Outline

- ✓ **Introduction**

- ✓ Design methodology

- **HW/SW co-design**

- ✓ Models of Computation

- Analysis
 - Synthesis