TKT-2431 Soc Design

Lec 3 – Platform-based design

Erno Salminen

Department of Computer Systems
Tampere University of Technology

Fall 2011
Copyright notice

- Part of the slides
  - adapted from slide set by Prof. Alberto Sangiovanni-Vincentelli
    - course EE249 at University of California, Berkeley
    - http://www-cad.eecs.berkeley.edu/~polis/class/lectures.shtml
- Part of figures from
Contents

- Orthogonalization (separation) of concerns
- Platform-based design
  - Concept of HW and SW platforms
  - Design-space exploration and mapping (ASV triangles)
- Case studies
  1. automotive
  2. wireless system
At first

Make sure that simple things work before even trying more complex ones
Intro [Keutzer] (1)

- Must balance production costs with development time and cost in view of performance and functionality considerations.

- Design time and cost dominate decision-making process:
  - Design reuse and early detection of non-feasible choices
  - Rapidly increasing NRE costs
  - Make parts that are likely to have high-volume production from a single mask set.

- Designs must be captured at the highest level of abstraction.

- The size of component that can be reliably designed in deep submicron technologies is in the range of 50k - 100k gates:
  - Many small components instead of few large
  - [Sylvester, Keutzer, IEEE Proc, Vol.89, Iss.4, Apr 2001]
Intro [Keutzer] (2)

- Common “hardware” denominator (hardware platform) can be shared across multiple applications in a given application domain,
  - Production volume increases
  - Overall costs may eventually be (much) lower than in the case of customized chips
  - Performance is ”close enough” to optimized custom chip

- Concurrency increases
  - Needs efficient design and implementation methods
  - Data movement and transformation is of central importance
  - Communication is too often intertwined with the behavior
    - Reuse difficult
    - E.g. cannot change the network or the processing element

- Separating communication and computation is essential for handling system design complexity
  - i.e orthogonality of concerns
Orthogonalization of concerns

- Means separation of the various aspects of design
  - More effective exploration of alternative solutions
  - Combine the locally optimum choices
- Example: the orthogonalization between functionality and timing exploited in the synchronous digital design methodology
  - Provided that propagation delays in combinational blocks are all within the clock cycle
  - Verifying the correct behavior is restricted to the functionality of the combinational blocks
  - Static timing analysis tools check the timing within single clock cycle (critical path analysis)
  - Major design speed-up factor versus the more liberal asynchronous design methodology
Orthogonalization of concerns (2)

The most important aspect of any design is how it is partitioned.
The second most important aspect of any design is its interfaces.

– M. Keating.
Design Y-chart

1. System Function
2. System Microarchitecture
3. Mapping
4. Function on Microarchitecture

Refine

Implementation of System

i.e. platform (instance)

Figure 1: Overall Organization of the Methodology
System function

- System implements a set of functions
  - Function is an abstract view of the behavior of an aspect of the system.
  - Set of functions is the input/output characterization of the system with respect to its environment
  - "What is the result if the input (sequence) is … ?"
  - NO notion of implementation associated with it

- Example:
  - **Function**: when the engine of a car starts (input), display the number of revolutions per minute of the engine (output)"
  - **Not a pure function**: “when the engine starts, display in digital form of the number of revolutions per minute on the LCD panel”
Function description languages

- Natural language is the most used
  - Finnish, English, figures, tables...

- Executable
  - Matlab, C, SDL, UML...

- Formal
  - Allow automatic transformations and property checking
  - Defined with a Model of Computation (MoC)
  - Not in scope of this course but in TKT-1527
    Digital system design issues
System (micro)architecture

- **Architecture** defines an interface specification
  - Independent of the actual implementation
- **Micro-architecture** defines how this functionality is actually realized as a composition of modules and components, along with their associated SW
  - E.g. LCD is a physical component of a micro-architecture
- Instruction-set of a CPU is *architecture*, whereas CPU’s RTL captures *micro-architecture*
- Distinction is mostly of academic interest
  - Term architecture is often used in both meanings
- Micro-architecture may be either completely or partly fixed
  - Fixed micro-arch simplifies design but limits optimality
- Examples of commercial platforms introduced in the course TKT-3547 SoC platforms
Mapping

- Functions are assigned (mapped) to the components of the micro-architecture
  - Some authors use term "allocation" for this
  - e.g. GUI is mapped to CPU, CRC to custom HW component and so on
- Based on estimates of cost and performance
Another way: Separating computation and communication

- Separated computation and communication:
  - Different teams design the processors and on-chip network

→ The separation may happen both ways
  1. comp/comm separated in behavior
  2. comp/comm separated in architecture

- Or vice versa
  1. separate behav./arch. of communication
  2. separate behav./arch. of computation
Separating comp/comm in function

- Regular code (intertwined computation and communication)
  
  ```
  for i=0:256 {
      data[i] = func(src [i])
      if (i%32 == 0)
          {for j=0:32 {...}}
  }
  for i=0:256 {
      ... 
      dst [i] = ...
  }
  ```

- Separated computation and communication

  ```
  data = read (&src[0], amount);
  uncompress_data (data);
  calculate_fdg (data);
  send (&dst[0], data);
  ```

(*) Private memory is cache in the first example, and scratch-pad memory in the second. In the 2nd case, src and dst can local memories of toher CPUs as well.
Separating comp/comm in arch

Note! Interface between resources and network should be standardized.
Design space exploration

- Seeks to optimize system parameters
  - HW allocation and frequencies, task mapping, scheduling, NoC parameters
  - Architecture exploration is sometimes used as synonym

- Very large design space, i.e. number of parameter combinations
  - Exploration must be automated
  - Abstract models and heuristic optimization algorithms utilized

- Design space is first pruned, reduced, with coarse models
- Select promising solutions for more detailed analysis
Platform-based design
ASV: The Next Level of Abstraction in the Architecture Space

ASV = Alberto Sangiovanni-Vincentelli

IP Block Performance
Inter IP Communication Performance Models

RTL

SDF
Wire Load

Gate Level Model
Capacity Load

Transistor Model
Capacity Load

…”new methodologies that emphasize re-use at all levels of abstraction are a “must”, and this is a major focus of our work…” [Keutzer et al.]
Hardware Platform is a family of architectures that satisfy a set of architectural constraints imposed to allow the reuse of hardware and software components.

A platform instance is a set of components that is selected from the library (the platform) and whose parameters are set.

Basic PC (x86 ISA, standard buses etc) is one example in GPP domain

Texas Instruments OMAP for wireless handsets
HW platform (instance)

- A set of interconnected components
- Trade-off between
  - Supported application space, i.e. flexibility
  - Size of micro-architecture space that satisfies HW platform definition

a) HW platform (or template)

b) Modified platform (i.e. platform instance)

[Pao-Ann Hsiung, SoC Design Flow & Tools course slides, 2003]
Hardware Platforms Not Enough!

- Hardware platform has to be “extended” upwards to be really effective in time-to-market
- Interface to the application software is an Application Program Interface (API)
  - Also called Hardware-Dependent SW (HdS)
  - Better SW reuse
- Software layer performs abstraction:
  - Programmable cores and memory subsystem with (RT)OS
  - I/O subsystem via Device Drivers
  - SW platform
Software Platforms

- Platform approach
  - Application SW
    - Device drivers
      - RTOS
      - Compilers
      - Device drivers
      - BIOS
  - Network communication
- Trad. approach
  - Application SW
  - Hardware platform
    - Input devices
    - Output devices
ASV Triangles

All possible applications

Application space covered by Z

Application Instance A

All possible platforms

(These 2 cannot implement A)

Mapping application onto platform

Selected system platform Z

Architecture Exploration

Design-Space Exploration

Selected Platform Instance

Architectural space of Z
ASV Platforms

- The design process is meet-in-the-middle:
  - **Top-down**: map an instance of the top platform into an instance of the lower platform and propagate constraints
  - **Bottom-up**: build a platform by defining the “library” that characterizes it and a performance abstraction (e.g., number of literals for tech, Independent optimization, area and propagation delay for a cell in a standard cell library)

- The library has elements and interconnects

For every platform, there is a view that is used to map the upper layers of abstraction into the platform and a view that is used to define the class of lower level abstractions implied by the platform.
A Discipline of Platform-Based Design

Application

Programming Model: Models/Estimators

Kernels/Benchmarks

Architecture(s)

Architectural Platform

Microarchitecture(s)

Cycle-speed, power, area

Functional Blocks, Interconnect

Circuit Fabric(s)

Silicon Implementation Platform

Manufacturing Interface

Delay, variation, SPICE models

Basic device & interconnect structures

Silicon Implementation
Platform-Based Implementation

- Platforms eliminate large loop iterations for affordable design. Infeasibility detected earlier
- Restrict design space via new forms of regularity and structure that surrender some design potential for lower cost and first-pass success
- The number and location of intermediate platforms is the essence of platform-based design
Platform-Based Design Process

- Different situations will employ different intermediate platforms, hence different layers of regularity and design-space constraints

- Critical step is defining intermediate platforms to support:
  - **Predictability**: abstraction to facilitate higher-level optimization
  - **Verifiability**: ability to ensure correctness

- Skipping platforms can potentially produce a superior design by enlarging design space – if design-time and product volume ($) permits

- However, even for a large-step-across-platform flow there is a benefit to having a lower-bound on what is achievable from predictable flow
Design methodology (1)

- The ultimate goal in this regard is to create a library of functions
  - Can be used for all new designs
  - Plug-and-play integration
  - Functions are associated with HW and SW implementations
  - Implementation depends on choice of component at the architecture platform level

- Multiple levels of functionality supported in such a library
  - Lower levels that are closer to the physical implementation change because of the advances in technology
  - While the higher levels tend to be stable across product versions
Design methodology (2)

- Function/Architecture co-design happens at all levels of abstractions
- Each platform is an “architecture” since it is a library of usable components and interconnects
  - Can be designed independently of a particular behavior.
- Usable components can be considered as “containers”, i.e., they can support a set of behaviors
- Mapping chooses one such behavior. A **Platform Instance** is a mapped behavior onto a platform
- A fixed architecture with a programmable processor is a platform in this sense. A processor is indeed a collection of possible behaviors
- A SW implementation on a fixed architecture is a platform instance.
### Rough comparison of IC technologies w.r.t integrator/system house

<table>
<thead>
<tr>
<th>Parameter</th>
<th>FPGA</th>
<th>Platform chip</th>
<th>ASIC</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAD tool cost</td>
<td>$2000</td>
<td>? ($thousands?)</td>
<td>$Millions</td>
</tr>
<tr>
<td>Mask cost</td>
<td>0</td>
<td>0</td>
<td>$1.4M @90nm</td>
</tr>
<tr>
<td>Bug fix</td>
<td>1 hours</td>
<td>&lt;1 hour</td>
<td>~10 weeks</td>
</tr>
<tr>
<td>Electrical and optical check % debug</td>
<td>Vendor’s problem</td>
<td>Vendor’s problem</td>
<td>Your problem!</td>
</tr>
<tr>
<td>Time to market</td>
<td>Quite fast</td>
<td>Fastest</td>
<td>Slow</td>
</tr>
<tr>
<td>Die size</td>
<td>20x</td>
<td>5x</td>
<td>1x</td>
</tr>
<tr>
<td>Volume cost</td>
<td>1x – 20x</td>
<td>2-4x</td>
<td>1x</td>
</tr>
<tr>
<td>Speed</td>
<td>0.3x</td>
<td>0.8x</td>
<td>1x</td>
</tr>
<tr>
<td>Power</td>
<td>5x</td>
<td>2x</td>
<td>1x</td>
</tr>
</tbody>
</table>

Original table from [“FPGAs and Structured ASICs - Overview & Research Challenges”, Vaughn Betz, Altera Corporation]

Case studies: automotive and wireless system
Case 1: Automotive

- Magneti–Marelli Automotive Engine Control
- Strong control component
- Tight safety constraints

Components

- Failure detection and recovery of input sensors
- Computation of engine phase, status and angle, crankshaft revolution speed and acceleration
- Injection and ignition control law
- Injection and ignition actuation drivers
Case 1: Function

- Application had a large part of legacy design
  - Existing implementation had 135,000 lines of C source code not including comments
- Exact functionality had to be extracted
  - Formal co-design FSM (CFSM) representation
  - 89 CFSMs and 56 timers
  - Behavior of actuators and part of sensors rewritten in formal model
- Redesign using layered SW structure
  - 3 CPUs and 2 SW partitions could be tested
  - 3 different IO subsystems
  - design space exploration
Case 1: Results

- Performance estimations in Cadence VCC
  - Error only 11% w.r.t real HW
- Large (>86%) SW reuse
- Added another CPU since it takes only 4-6% of area
  - Area dominated by flash memory
Case 2: Wireless system design

- Sufficient flexibility that the network node can support
  - various system configurations
  - communication requirements
  - radio front ends
- Ad-hoc network
  - Topology may vary rapidly and unpredictably
- Overall system power consumption as well as the power consumption of an individual network node is minimized (100 uW)
- Designed with CFSMs
Case 2: Protocol stack

TABLE I
COMPLEXITY OF THE INTERCOM SPECIFICATIONS (USING CFSMs)

<table>
<thead>
<tr>
<th>Layer</th>
<th>Blocks</th>
<th>C-code (lines)</th>
<th>State-transition Diagram (states)</th>
</tr>
</thead>
<tbody>
<tr>
<td>User interface</td>
<td>1</td>
<td>100</td>
<td>-</td>
</tr>
<tr>
<td>Mulaw</td>
<td>2</td>
<td>100</td>
<td>-</td>
</tr>
<tr>
<td>Transport</td>
<td>5</td>
<td>300</td>
<td>-</td>
</tr>
<tr>
<td>MAC</td>
<td>23</td>
<td>270</td>
<td>42</td>
</tr>
<tr>
<td>Transmit</td>
<td>6</td>
<td>120</td>
<td>16</td>
</tr>
<tr>
<td>Receive</td>
<td>6</td>
<td>140</td>
<td>2</td>
</tr>
<tr>
<td>Synchronization</td>
<td>3</td>
<td>-</td>
<td>17</td>
</tr>
<tr>
<td>Total</td>
<td>45</td>
<td>1030</td>
<td>77</td>
</tr>
</tbody>
</table>

“small-footprint” RTOS adds 200 lines of C

Traditional C code (without CFSM) 15 000+ 13 000 lines of C
Case 2: HW platform (1)

- Platforms combine
  - Different levels of data granularity (word-versus bit-oriented)
  - Computation rates (in terms of clock frequencies)
  - Flexibility (programmable processor, configurable logic, and hardwired)

- ‘Too General’ platform may lead to overspecification
  - Excess area and power overhead
  - Platforms can be tuned to certain application

Fig. 7. Generic intercom platform.
Case 2: HW platform (2)

Platforms differ from each other in terms of
1. Processor choice
   - ARM: popular RISC
   - Xtensa: extensible CPU (more info on lec 9)
2. Clock speeds 1-200 MHz
3. Amount of FPGA, fixed logic, and memory

Each architectural module is represented by a
1. functional model
2. first-order performance, energy, and area estimation models.

Fig. 7. Generic intercom platform.
Case 2: Results (1)

- Different CPUs and mappings were tried out.
- Merging the Mulaw computation with the transport and user interface functions on the CPU @ 11 MHZ.
  - Dramatic RTOS overhead due to the excessive context switches.
  - Even processor @ 200 MHz (and expending dramatic levels of power dissipation), a pure SW solution is not feasible.
Case 2: Results (2)

Figure 9. Platform Exploration and Partitioning Results.

Layer
- Synchronization
- Transmit
- Receive
- MAC
- Mulaw
- Inv.Mulaw
- Transport
- User Interface

% Processor utilization
- RTOS overhead

always on HW

HW or SW

always on SW

UI + Tx + RTOS
UI + Tx + Mulaw + RTOS
UI + Tx + Mulaw + MAC + RTOS
Case 2: Results (3)

- SW performance estimates
  - Statistical cycles/instruction
  - CFSM analysis and data profiling
  - Error <15%

- Description with CFSMs very efficient

Figure 8. Performance Modeling of the Xtensa Processor.
Case 2: Results (4)

- Bursty nature of the protocol processor
  - Clock speed is determined by the peak computational requirements
  - Low processor utilization for each of the mappings
  - Energy reductions through power-down and dynamic voltage scaling possible

- First-generation PicoRadios
  - Xtensa processor running at 11 MHz (30 000 gates)
  - 50 kByte of SRAM
  - 10 000 equivalent FPGA gates
  - At most 10 000 gates ASIC block
    - Invariant function of the physical layer

- The estimated power dissipation is around 5mW
Conclusion

- Orthogonalization
  - Separate function and architecture
    - Functions have no assumptions on implementation (whether it is SW or HW)
    - Functions are mapped onto architecture
  - Separate computation and communication
  - Parts can be designed and optimized concurrently, smaller design space
  - However, design teams must not be orthogonalized and separated

- Same HW (platform) can be used in several products
- Also parts of application SW should be reused
  - HW must be abstracted (API, HdS)
- High-level models used for estimation
There are less free parameters when going down, and hence less possible architectures where one can end to. Make big decisions first.
Large steps between platforms

- The larger the step across platforms, the more difficult to:
  - predict performance
  - optimize at system level
  - provide a tight lower bound
  - achieve fast iteration loop

- Achievable design space may actually be smaller than with smaller steps
  - it is more difficult to explore and restriction on search impedes complete design space exploration

- The predictions/abstractions may be so wrong that design optimizations are misguided and the lower bounds are incorrect!

Erno Salminen - Sep. 2011
MESCAL flow

1. Designer creates *initial* micro-architecture
2. Compiler and simulator are *retargeted* to that micro-arch
3. Designer receives feedback on performance and cost
   - New iteration starts if required
   - Programmer’s view (SW API) remains the same

---

![Diagram](image)

**Fig. 14.** MESCAL design methodology.
Terminology definition

- In this course, terminology follows [Kangas, Phd thesis, TUT 2006]

- **Application**:  
  - An application is the functionality defined in the system specification.

- **Application process**:  
  - A behavioral entity in an application, i.e. the application consists of a set of processes. *Application task* is used as a synonym in this context.

- **Architecture exploration**:  
  - The architecture exploration can be divided into *component allocation*, *task mapping*, and *scheduling*. Allocation denotes the selection of processing elements and communication network. The application tasks are then mapped onto the allocated processing elements. Scheduling is used to define the order and timing of the execution of tasks and communication.

- **Platform-based design**:  
  - In platform-based design, successive refinements of specifications meet with abstractions of potential implementations [111]. The refinement and abstraction processes take place in layers that are called *platforms*.

- **Platform**:  
  - A set of libraries that include reusable and domain-specific models of both software and hardware components to be used in the system refinement as well as supporting models and descriptions for design automation [111]. For a component there can be several models with different abstractions in the library; e.g. a coarse performance estimate for early exploration and an accurate RTL model for cycleaccurate simulation and synthesis.

- **Architecture**:  
  - An instance of a platform. A set of hardware processing elements (PEs), memories, and their connections. These components can be instantiated from a library or described from scratch.
Terminology definition (2)

- [Kangas, Phd thesis, TUT 2006]

**Design space:**
- All the possible platform instances and mappings. With design space exploration, all or a portion of these combinations are analyzed with respect to an objective that is defined with a cost function. Design space exploration can be roughly divided into application and architecture exploration.

**Cost, performance:**
- The goodness of the platform instance is evaluated with a cost function. Cost (area, runtime, power) is to be minimized, performance (throughput, efficiency, frame rate) maximized

**System-level:**
- The system design can be roughly divided into system and component levels. System-level design deals with processing elements and their connections whereas component-level design refers to internal design and optimization of processing elements and communication networks. In this paper the term “highlevel” is used with the same meaning as “system-level”.

**IP:**
- Intellectual Property. A reusable, flexible, pre-verified, and well-documented design block with well-defined functionality and interface. Can be either software or hardware.

**Model:**
- This term has two distinct meanings. First, it denotes an instance of a system or part of it, e.g. UML model of an encryption component. Second, it refers to the modeling concepts (i.e. metamodel or MoC) behind a concrete realization, for instance a finite-state machine (FSM) [51].

**MoC:**
- Kienhuis et al. [58] define a model of computation as a formal representation of the operational semantics of networks of functional blocks describing computations. According to Lee et al. [73], MoC is a set of rules that provide a framework within which a designer builds models. A set of rules that govern the interaction of components is called the semantics of the model of computation.
<table>
<thead>
<tr>
<th>Parameter</th>
<th>FPGA</th>
<th>Standard Cell</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAD tool Cost</td>
<td>$2000</td>
<td>$Millions</td>
</tr>
<tr>
<td>Mask Cost</td>
<td>0</td>
<td>$1.4M US @ 90 nm</td>
</tr>
<tr>
<td>Bug Fix</td>
<td>1 hour</td>
<td>~10 weeks</td>
</tr>
<tr>
<td>Electrical &amp; Optical Check &amp; Debug</td>
<td>Vendor’s Problem</td>
<td>Your Problem!</td>
</tr>
<tr>
<td>Time to Market</td>
<td>Fast</td>
<td>Slow</td>
</tr>
<tr>
<td>Die Size</td>
<td>2X to 20X</td>
<td>1X</td>
</tr>
<tr>
<td>Volume Cost</td>
<td>1X to 20X</td>
<td>1X</td>
</tr>
<tr>
<td>Speed</td>
<td>0.3X to 0.6X</td>
<td>1X</td>
</tr>
<tr>
<td>Power</td>
<td>2X to 5X</td>
<td>1X</td>
</tr>
</tbody>
</table>

“FPGAs and Structured ASICs - Overview & Research Challenges”, Vaughn Betz, Altera Corporation
Ability to upgrade should increase income

E.g. WLAN adapter: updates enable new standards

Shorter time-to-market gives better profits

Updates extend lifecycle

reconfigurable product with download

revenue [€] / month

ASIC product

update 1

update 2

time [month]

[T.Kean, It's FPL, Jim - But Not as We Know It! Opportunities for the New Commercial Architectures], LNCS 1896, Springer-Verlag, 2000]