MPSOC Architecture Modeling

R. Ernst
TU Braunschweig

Overview

- introduction – MPSOC architecture trends
- implementation languages and architectures
- architecture model applications
- models for formal design methods
- summary
MPSOC architecture - trends

- system function integration
  - reactive and transformative parts
  - function IP, legacy code, new functions
- component and subsystem reuse (IP)
  - increased design productivity and reduced development cost
- programmable platforms
  - improved design productivity
  - increased volume
  - examples: network processors, multi-media platforms, automotive platforms, game platforms

MPSOC architecture - challenges

- design specialization
  - increased performance
  - reduced power consumption
  - lower cost and size
- design flexibility
  - late changes, platforms, reuse
- HW and SW IP integration
  - result of reuse
⇒ MPSOC architectures are heterogeneous
MPSOC architectures are heterogeneous

- different processing element types
  - processors, weakly programmable coprocessors, IP components
- different interconnection networks and communication protocols
- different memory types
- different scheduling and synchronization strategies

Example: Configurable platform (Nexperia™)

- Nexperia™ DVP hardware architecture (source: Th. Claasen, Philips, DAC 2000)

General Purpose RISC Processor
- 50 to 300+ MHz
- 32-bit or 64-bit

Library of Device Blocks
- Image coprocessors
- DSPs
- UART
- 1394
- USB
- ...and more

MIPS™ SDRAM TriMedia™

VLIW Media Processor:
- 100 to 300+ MHz
- 32-bit or 64-bit

Nexperia System Busses
- PI bus
- Memory bus
- 32-128 bit
Nexperia example: Viper Setup Box

- External SDRAM
- Interrupt controller
- Enhanced JTAG
- Universal async. receiver/transmitter (UART)
- ISO UART
- Reset
- Clocks
- IC debug
- CPU debug
- Universal serial bus
- Universal async. receiver/transmitter (UART)
- ISO UART
- Reset
- MIPS (PR3940) CPU
- MIPS bridge
- MIPS C-Bridge
- Memory controller
- TriMedia (TM32) CPU
- TriMedia C-Bridge
- MPEG-2 video decoder
- Adv. image composition Processor
- Video input processor
- Memory-based scaler
- MPEG system proc.
- ISO UART
- Fast PI bus
- Fast C-Bridge
- MIB
- C-bridge
- MIB
- PC
- IEEE 1394 link layer controller
- CRC
- DMA
- MPEG-2 Video decoder
- Video input processor
- Memory-based scaler
- MPEG system proc.
- Interrupt ctrl.
- Audio I/O
- Sony Philips Digital I/O
- Transport stream DMA
- General-purpose I/O
- Synchronous serial interface

VIPER chip layout - reuse and integrate

- VIPER Hardmacros (supplied)
- VIPER adaptable softmacros „Chiplets“
Managing HW architecture complexity

- development of application programmer interfaces (API) to hide complexity from application programmer and improve portability
- specialized RTOS to control resource sharing and interfaces

⇒ complex multi-level HW/SW architecture

Software architecture example

- layered software architecture with HW dependent SW and API

⇒ MPSOC SW is heterogeneous
### Application & Architecture

**Application:**
"Function"

**System View:**
- Application layer
  - Development
  - Implementation language
  - Architecture layer
  - Implementation

**Target HW/SW Architecture:**
- CoP
  - M
  - IP
  - DSP
  - MP
  - RTOS
  - I/O
  - Int
  - Bus
  - CTRL

**Implementation Language Semantics:**
- System of (communicating) processes
- Shared Memory
  - "g = a * c;"
  - "a = b + c;"
- Message Passing
  - "receive (a);"
  - "send (a);"

**Languages:**
- C, C++, Java, (SystemC)
- VHDL, SystemC, SpecC

© R. Ernst, TU Braunschweig
MPSOC 2003
Function Design → Implementation Language

- system level language: SDL, Simulink, Lustre, ....
- implementation language: manual implementation / code generation
- examples: • Real-Time Workshop • Tau

Diagram:
- "g = a * c;"
- "a = b + c;"
- shared memory
- "send (a);"
- "receive (a);"
- Processes P1, P2, P3, P4

"Lossy" function translation

- information lost in transformation
  - state dependent process behavior
    - hidden in processes
  - process coordination
    - expresses dependency and activation rules
    - important for efficient HW/SW architecture implementation
- example: data flow semantics → RTOS process activation
Token-to-event translation problem

- Application view
  - complex activation dependencies
  - no timing (partial order)

What is an activating event?

→ need transformation

- Scheduling analysis view
  - simple activation dependencies (e.g. task graphs)
  - event timing

Coordination must be kept in translation

- solution for example e.g. using token arrival curves (Jersak/Ernst, DAC 03)
- semantic preserving models needed to keep information
  - Metropolis, Funstate, SPI
Resource sharing

- HW/SW resources are shared
  - several SW processes mapped to one processing element
    ⇒ task scheduling
  - mapping several communications mapped to one communication path
    ⇒ communication (bus) scheduling
  - several process data mapped to one memory
    ⇒ memory assignment (space & time)
Resource sharing - 2

- resource sharing strategies
  - in time
    - execution sequence -> scheduling
  - in space
    - memory assignment
    - bus wire assignment

Architecture modeling applications

- implementation verification
- performance validation
  - response time
  - throughput (process exec/time unit)
  - bottleneck detection
- design optimization
  - design space exploration
  - power optimization
- cost determination (not this lecture)
MPSOC architecture modeling requirements

- given
  - an application and its environment modeled by a set of communicating processes
  - a heterogeneous HW/SW target architecture
  - an implementation of the processes on the architecture

- model
  - the HW/SW architecture information flow
  - system timing (and power consumption)

Modeling Challenges

- model complexity
  - HW/SW system state space
  - simulation run-times and analysis complexity
  - model abstraction

- activation modeling
  - simulation pattern development
  - environment modeling

- complex non-functional interdependencies
  - shared communication
  - shared components
  - shared memory
Complex non-functional interdependencies

- resource sharing introduces complex non-functional interdependencies ("cross talk")

Interdependency example

- anomalies: best case can become worst case
Modeling Challenges - cont’d

- complex design objectives and constraints

Reaction time of airbag after crash?

\[ t_{\text{crash}} + t_{\text{sens}} + t_{\text{detc}} + t_{\text{bus}} + t_{\text{ctrl}} + t_{\text{act}} + t_{\text{act}} + t_{\text{airbag}} \]

\[ = t_{\text{com}} + t_{\text{API}} + t_{\text{drive}} + t_{\text{API}} + t_{\text{API}} + t_{\text{drive}} + t_{\text{API}} + t_{\text{API}} \]

\[ = t_{\text{com}} + t_{\text{API}} + t_{\text{drive}} + t_{\text{API}} + t_{\text{API}} + t_{\text{drive}} + t_{\text{API}} + t_{\text{API}} \]

\[ = t_{\text{com}} + t_{\text{API}} + t_{\text{drive}} + t_{\text{API}} + t_{\text{API}} + t_{\text{drive}} + t_{\text{API}} + t_{\text{API}} \]

\[ = t_{\text{com}} + t_{\text{API}} + t_{\text{drive}} + t_{\text{API}} + t_{\text{API}} + t_{\text{drive}} + t_{\text{API}} + t_{\text{API}} \]

MPSoC modeling - goals

- current goal: Target architecture co-simulation
  - supports system-level validation
  - uses library of component and communication models
  - requires executable code and software platform
  - extensive simulation required for complex MPSoC or distributed systems

- research goal: support formal methods for design space exploration, system-level optimization and analysis
  - different modeling approach required
Architecture model structure

- model type
  - system model

- influenced by
  - communication pattern
  - shared memory access
  - environment model

- activation

- component & communication execution model
  - resource sharing strategy
  - process activation
  - component state (caches)

- process execution model
  - execution path – data dependent
  - path execution – arch. dependent
  - communication – data & arch. dep.

Process execution model

- timing and communication depend on
  - execution path
  - architecture
  - communication mechanism and volume

({b1, …, bn} basic blocks)
Process timing and communication

- Process timing and communication can be evaluated by
  - Simulation/performance monitoring
    - Trigger points at process beginning and end
    - Stimuli required, e.g. from component design
    - Data dependent execution → upper and lower timing bounds
  - Simulation challenges
    - Coverage?
    - Cache and context switch overhead due to run-time scheduling with process preemptions
    - Influence of run-time scheduling depending on external event timing
  - Formal analysis of individual process timing
    - Serious progress in recent years
  - Discussion see book chapter and literature

Architecture model structure

- Model type
  - System model
  - Influenced by
    - Communication pattern
    - Shared memory access
    - Environment model

- Component & communication execution model
  - Resource sharing strategy
  - Process activation
  - Component state (caches)

- Process execution model
  - Execution path – data dependent
  - Path execution – arch. dependent
  - Communication – data & arch. dep.
Component and communication execution model

- Resource sharing strategy
  - process and communication scheduling
  - static execution order
  - time driven scheduling
    - fixed
    - dynamic
  - priority driven scheduling
    - static priority assignment
    - dynamic priority assignment

- timing depends on environment model
  - frequency of process activations or communication
  - solution for activation transformation proposed in (Jersak/Ernst DAC 03)

Ex 1: Time driven scheduling

- time division multiple access (TDMA)
  - periodic assignment of fixed time slots
  - applicable to pe or ce

![TDMA example image]
TDMA

- predictable and independent performance down scaling allows to merge individual solutions
  \[ t_{peTDMA}(P_i, pe_i) = \left\lfloor \frac{t_{pu}(P_i, pe_i) - t_{sw}}{t_{Pu}} \right\rfloor \cdot t_{puTDMA} + t_{pu}(P_i, pe_i) \mod t_{pi} \]

- time slot size adaptable to different service levels
- generates output jitter as a result of execution times
- problems
  - utilization
  - extended deadlines

Ex 2: Static priority with arbitrary deadlines

- complex execution sequence - may create output bursts
- found in communication scheduling and multiprocessing

Analysis solution e.g. by Lehoczky
RTOS Overhead

Example: Static priority scheduling (ERCOSEK™)

RTOS overhead increases response times

RTOS and scheduling effects combined

© R. Ernst, TU Braunschweig MPSOC 2003
Cache effects

- cache contents replaced by other processes
  - increased execution time
  - must be considered in analysis
- scratch pad memories as alternative

System model

- static priority scheduling
- FCFS scheduling
- TDMA scheduling
- proprietary (abstract info)
- earliest deadline first scheduling
- static execution order scheduling
Component coupling

- independently scheduled subsystems are coupled by data flow

⇒ subsystems coupled by stream of events
⇒ coupling corresponds to event propagation

Architecture model structure

- model type
  - system model
  - component & communication execution model
  - process execution model

- influenced by
  - communication pattern
  - shared memory access
  - environment model
  - resource sharing strategy
  - process activation
  - component state (caches)
  - execution path – data dependent
  - path execution – arch. dependent
  - communication – data & arch. dep.
System timing analysis with event propagation

- analysis scope extension to several subsystems
  - holistic approach, e.g. Tindell and Pop/Eles
  - used for automotive software
- event model generalization for a set of scheduling strategies
  - arrival and service curves Chakratborty/Thiele
  - new analysis approaches needed, e.g. Baruah
  - used for network processor design
- event stream model adaptation
  - use abstract interface stream properties to couple local analysis
  - used e.g. for automotive software

Event stream models

- periodic events
  \[ t_{\theta 1} \rightarrow t_p \rightarrow t_{\theta 2} \rightarrow t_p \rightarrow t_{\theta 3} \rightarrow \cdots \]
  \[ t_{\theta 1} - t_{\theta 2} = t_p \]
  \[ t_{\theta i+1} - t_{\theta i} = t_p \]
  
- periodic events with jitter
  \[ t_{\theta 1} \rightarrow t_p - j/2 \rightarrow t_{\theta 2} \rightarrow t_p - j/2 \rightarrow t_{\theta 3} \rightarrow \cdots \]
  \[ t_p - j \leq t_{\epsilon i+1} - t_{\epsilon i} \leq t_p + j ; \]
  \[ j < t_p \]

- events with minimum inter arrival times
  - burst events, packets, sporadic events, etc.
  \[ t_{\min} \rightarrow t_{\min} \rightarrow \cdots \rightarrow t_{\min+1} \rightarrow \cdots \]
  \[ t_p - j \leq t_{\epsilon i+1} - t_{\epsilon i} \leq t_p + j ; \]
  \[ t_{\epsilon i+1} - t_{\epsilon i} \geq t_{\min} ; \]
  \[ t_{\epsilon i+1} - t_{\epsilon i} \geq t_{\min} \]
Event stream example

**Subsystem 2**
- **Sens**
- **CPU**
- **M1**
- **HW**

**Subsystem 1**
- **IP1**
- **M3**
- **IP2**
- **DSP**
- **M2**

Event propagation and analysis principle

1. Environment model
2. Local analysis
3. Derive output event model
4. Transform to expected input event model
5. Until convergence or non-schedulability
Example revisited

non-functional dependency cycle

Dependency cycle

resynchronization reduces jitter
Some open issues

- system mode dependent timing
  - different load situations and response time requirements
  - tagged tokens -> tagged event streams
- (complex) hierarchical process and communication scheduling
- global scheduling optimization

Architecture model summary

- current MPSOC architecture models are primarily used for simulation
- growing complexity suggests formal methods for optimization and analysis
- emerging formal approaches supported by multi-level architecture modeling
- abstract event flow model enables heterogeneous system analysis
Literature

- see: www.spi-project.org

Acknowledgement

- The following persons contributed in developing these slides
  - Bettina Boettger
  - Jörn Braam
  - Marek Jersak
  - Kai Richter