

Predictability in the CoMPSoC platform

Benny Åkesson



TU / **e**

Technische Universiteit
Eindhoven
University of Technology

Where innovation starts

Trends in MPSoC Design

- ▶ Embedded system design gets increasingly complex
 - Moore's law allows increased component integration
 - Digital convergence creates a market for highly integrated devices
- ▶ Systems are implemented as MPSoC platforms with
 - a large number of heterogeneous intellectual property (IP) components
 - many concurrently executing applications with real-time requirements
- ▶ Pressure to **quickly design systems** in a **cost-effective** manner



Verification Problem

- ▶ Resource sharing
 - is required to reduce cost
 - introduces **interference** between sharing applications
 - makes timing behavior inter-dependent
- ▶ Verification is a design bottle-neck
 - Verification is often done by simulation of use-cases executing on platform
 - Number of use-cases **grows exponentially** with the number of applications
 - System-level simulation is slow, resulting in poor coverage
 - Reverification required if an application is added or changes behavior
- ▶ Verification is **costly** and effort is expected to **increase** in future!



Formal Verification

- ▶ **Formal verification** is an alternative to simulation
 - Provides analytical bounds on latency or throughput
 - Verification with mathematical proofs instead of slow simulation
 - Covers all inputs and combinations of running applications

- ▶ Formal verification requires **predictable systems**
 - Requires timing models of hardware, software, and mapping
 - Most industrial systems are not designed with formal analysis in mind

Problem Statement

- ▶ Current trends make it **increasingly difficult to verify** real-time systems with fast time to market
- ▶ We require a **predictable system** (hardware + software + mapping tool) that **enables formal verification** of real-time applications
- ▶ The contributions of this presentation are:
 - An **introduction** to a predictable MPSoC platform
 - **Three general techniques** for predictable system design
 - **Example uses of techniques** in different resources
 - Processor tiles, interconnect, and memory tiles

Presentation Outline

Introduction

CoMPSoC overview

Memory tile

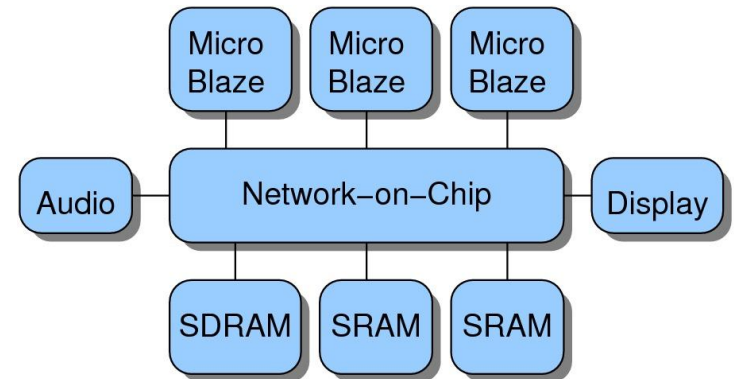
Interconnect

Processing tile

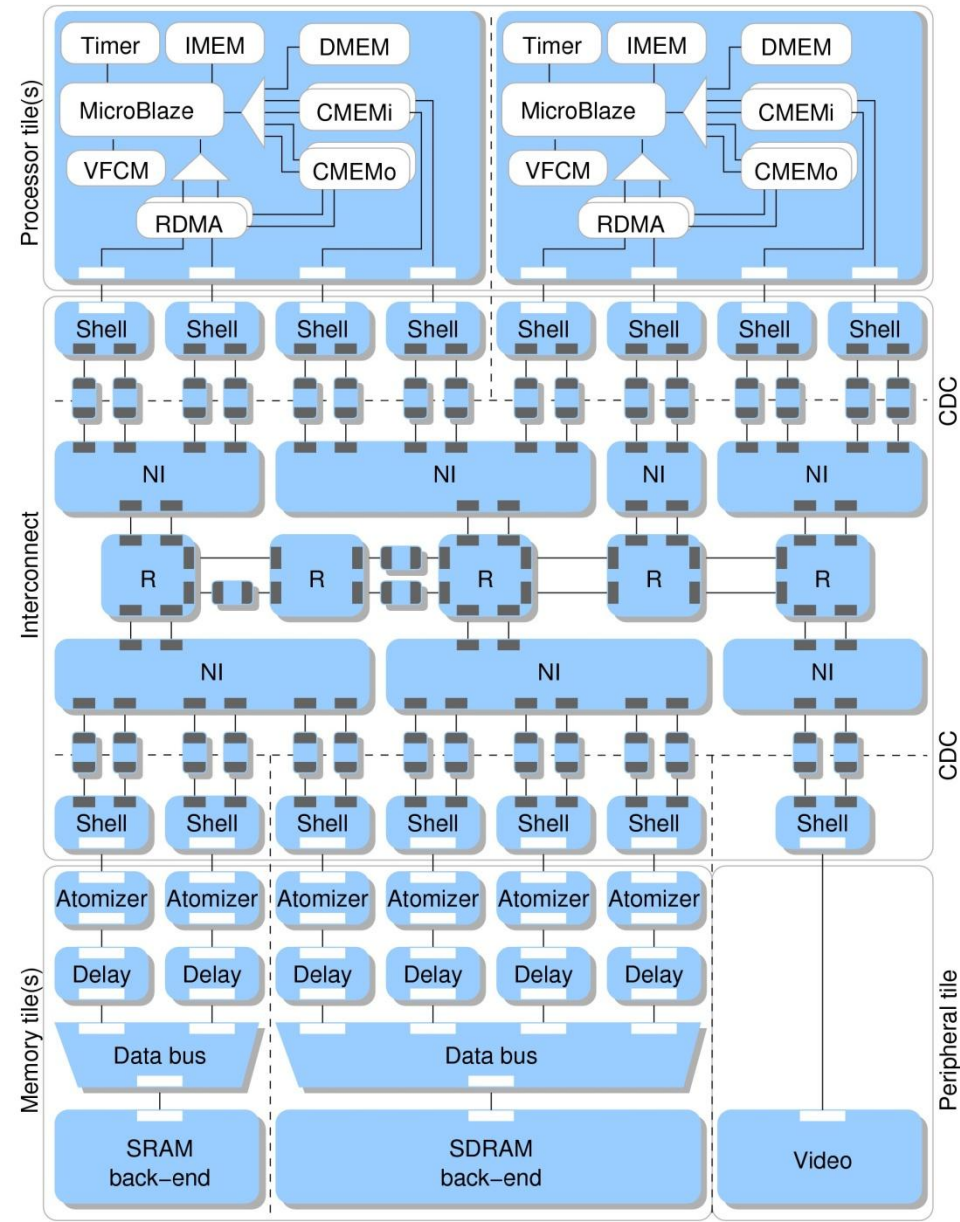
Conclusions

CoMPSoC Overview

- ▶ **CoMPSoC** is a real-time multi-processor platform
 - Supports formal verification using data-flow analysis (predictability)
 - Provides complete temporal isolation between applications (composability)
- ▶ Components of **tiled architecture**
 - Processing tiles with MicroBlaze cores
 - \mathcal{A} ethereal network-on-chip
 - Memory tiles with SRAM or SDRAM
 - Peripheral tiles
- ▶ Platform implementations both in
 - SystemC for prototyping
 - VHDL for FPGA instance
- ▶ Platform supported by an **automated design flow**



The CoMPSoC Architecture



► Processing tile

- Instruction memory (IMEM)
- Data memory (DMEM)
- Incoming Communication Memory (CMEMi)
- Outgoing Communication Memory (CMEMo)
- Voltage and Frequency Control Module (VFCM)
- Remote DMA (RDMA)

► Interconnect

- Network Interface (NI)
- Router (R)
- Clock Domain Crossing (CDC)

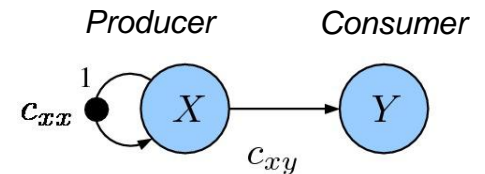
Application Model

- ▶ An **application** consists of a set of **tasks**
 - May be distributed over multiple processing tiles
 - Tasks may share a processing tile
 - Tasks communicate via distributed shared memory

- ▶ Applications have a **mixed time-criticality**

- **Firm real-time applications**

- E.g. streaming software-defined radio application
- Failure to satisfy requirement may violate correctness
- Modeled as data-flow graphs with latency and/or throughput requirements



- Soft real-time applications

- **Non-real-time applications**

- Any programming model
 - E.g. shared memory, KPN, or dynamic data-flow
- Not assumed to be well-specified
- No real-time requirements, but must be perceived as responsive

Predictable Systems

- ▶ **Execution time (ET)**
 - The time a request uses a resource before finishing
 - Execution of the **request on an unshared resource**
- ▶ **Response time (RT)**
 - Captures effects of **resource sharing**
 - Execution time plus interference from other requestors
- ▶ Predictable systems are built from predictable components
 - Systems with useful **worst-case execution time (WCET)** and **worst-case response time (WCRT)** for every request on every resource
 - E.g. for tasks on processors, and transactions in interconnect and memory

Enabling Formal Verification

- ▶ Benefits of separation of WCET and WCRT
 - Considers **independent analysis** of resource and arbiter
 - Analysis covers **all combinations** of predictable resources and arbiters
- ▶ Predictable systems **enable formal verification**
 - Application is a set of requestors for resources with bounded WCRT
 - WCRT of requests is used with performance analysis frameworks
 - E.g. Data-flow analysis, Real-time calculus, Network calculus

Presentation Outline

Introduction

CoMPSoC overview

Memory tile

Interconnect

Processing tile

Conclusions

Predictable SRAM Resource

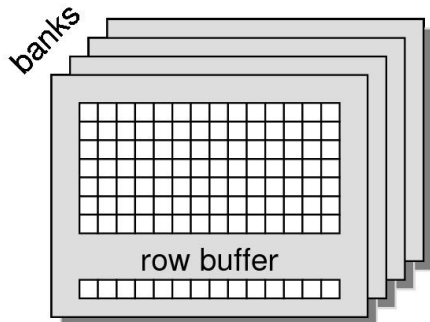
- ▶ Simple SRAMs read/write a random word in a single cycle
- ▶ **Requests may be large, resulting in long execution times**
 - **Atomizer** chops large requests into smaller requests (**atoms**) of fixed size
 - Small responses merged back to expected size
 - Makes all memory requests appear **homogeneous**
- ▶ **Execution time of memory depends on other resources**
 - Complicates analysis
 - **Make execution independent** by securing input data and output space
 - **Buffer atoms completely** before making them eligible for scheduling
 - Decouples memory from production of interconnect and processor
 - **Ensure sufficient space** for responses before scheduling atoms
 - Decouples memory from consumption of interconnect and processor

Predictable Shared SRAM resource

- ▶ We have arrived at a predictable SRAM resource
 - (Worst-case) execution time of a scheduled atom is constant
 - Next challenge is sharing the resource and bound response times
- ▶ **Some memory clients issue many requests, starving others**
 - Arbiter is unpredictable, such as FIFO and static-priority arbitration
 - Use a **budget-scheduler** (predictable through resource reservations)
 - E.g. Time-Division Multiplexing, Weighted Round Robin, Fair Queuing, and Credit-Controlled Static-Priority
 - Choose arbiter according to WCRT requirements of memory clients
- ▶ A predictable arbiter enables response time to be bounded
 - Arbiter bounds the **number of interfering atoms**
 - The WCET of SRAM atoms is known and constant, bounding interference
 - $WCRT = WCET + \text{bounded interference}$

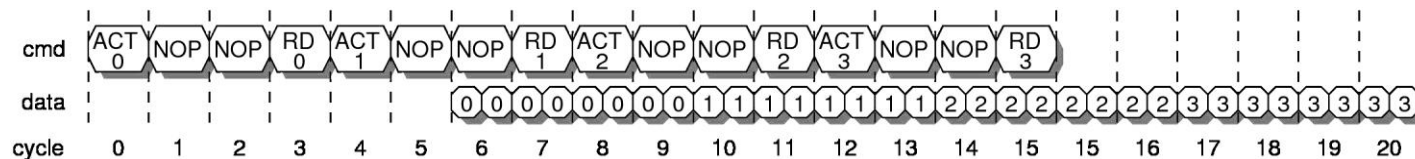
Problem with SDRAM

- ▶ SDRAM memories are more complicated than simple SRAMs
- ▶ The ET of an SDRAM request is **highly variable** and **traffic dependent**
 - Depends on if target row is open, if read or write, if time to refresh etc.
 - WCET is pessimistic and guaranteed bandwidth is very low
 - Less than 16% bandwidth can be guaranteed for all DDR3 devices
- ▶ SDRAM bandwidth is a **scarce resource** that must be efficiently used
 - Additional bandwidth cannot be added due to cost constraints



Predictable Unshared SDRAM Resource

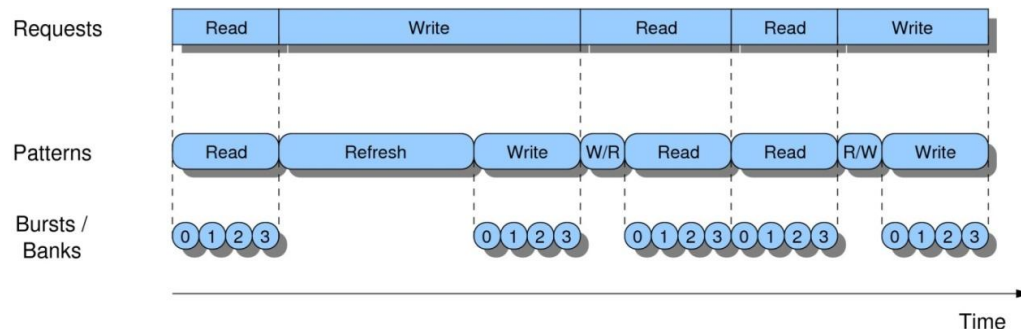
- ▶ **Restrict memory requests** to prevent inefficient accesses
 - Restricted to homogeneous atoms
 - Larger atoms to **amortize overhead** of read/write switching and row misses
 - Atoms are **served non-preemptively** due to high preemption cost
- ▶ Atoms are mapped to statically computed **memory patterns**
 - Sequences of SDRAM commands computed at design time
 - There are **six types** of patterns
 - Idle, read, write, r/w switch, w/r switch, and refresh patterns



Read pattern for DDR2-400

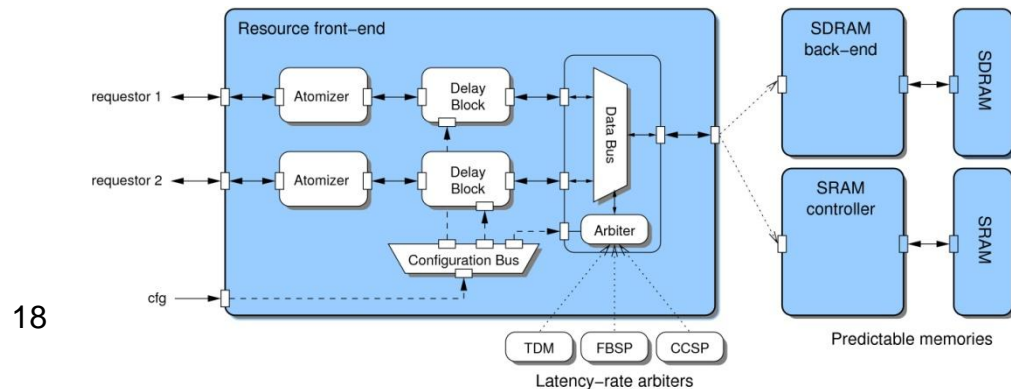
Memory Patterns

- ▶ Atom to pattern mapping:
 - No request → idle pattern
 - Read request → read pattern (potentially first w/r switch)
 - Write request → write pattern (potentially first r/w switch)
 - Refresh pattern issued periodically every 7.8 μs
- ▶ Patterns **abstract SDRAM command scheduling** to higher level
 - Patterns are easier to schedule and analyze, due to less dependencies
 - Moves intelligence from the memory controller to design-time tools



Predictable Shared SDRAM Resource

- ▶ Memory patterns **bound execution times and the guaranteed bandwidth**
 - Mapping determine worst-case atom to pattern mapping
 - ET and transferred data of patterns are known at design time
- ▶ Predictable arbitration **bounds response times**
 - Bounds the number of interfering atoms
 - Mapping determine worst-case atom to pattern mapping
 - ET of interfering patterns are known at design time
- ▶ For **any combination** of supported predictable memory and arbiter
 - SRAM, DDR2, DDR3, LPDDR, LPDDR2



Presentation Outline

Introduction

CoMPSoC overview

Memory tile

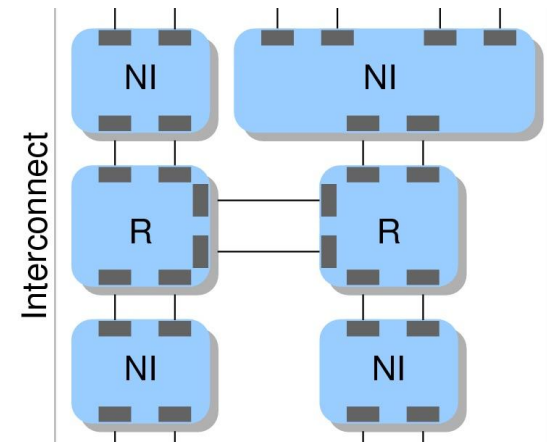
Interconnect

Processing tile

Conclusions

Interconnect Architecture

- ▶ Tiles are interconnected using a **network on chip**
- ▶ Main network architecture components
 - Network interface (NI)
 - (de)packetizes data
 - buffers requests until they can enter/exit the network
 - Router (R)
 - Forwards packets



Predictable Interconnect

- ▶ Distributed resource with **several arbitration points**
 - Predictable arbitration per router
 - Packets atomized into **flits** of three words for fine-grained scheduling
- ▶ **RT is very high if worst-case interference in every arbiter**
 - Use **global arbitration** to synchronize arbiters
 - Implemented using **contention-free pipelined TDM arbitration**
 - Global TDM schedule computed at design time
 - A flit in the network is **scheduled in consecutive TDM slots** in all routers
 - No buffering required in routers
 - All buffers are in network interface, which is cheaper
 - Requires sophisticated TDM allocation tool
 - Abstracts distributed resource to a **single pipelined resource**

Predictable Interconnect

- ▶ **A global notion of time is required to synchronize TDM slots**
 - Problem to implement in MPSoC based on a single clock
 - We implement notion of time in a **distributed fashion**
 - Everyone handshakes with neighbors before advancing to next slot
 - Suitable for asynchronous and mesochronous implementation
- ▶ **All NIs and routers need to store a TDM table for every link**
 - Path stored by NI in packet headers (source routing)
 - No arbitration logic required in routers, only forward packets
 - No contention possible due to global TDM arbitration
 - TDM tables hence only required in Nis
- ▶ The interconnect is **predictable**
 - The execution time of a flit is constant and three cycles
 - The minimum amount of data in a flit is bounded
 - The path of a packet (number of hops) is known at design time
 - The global TDM table is determined at design time
 - This enables the WCRT of the interconnect to be bounded

Presentation outline

Introduction

CoMPSoC overview

Memory tile

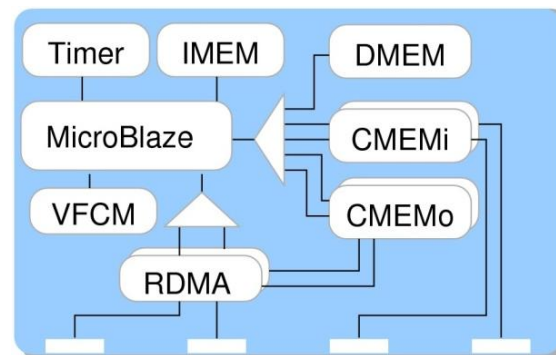
Interconnect

Processing tile

Conclusions

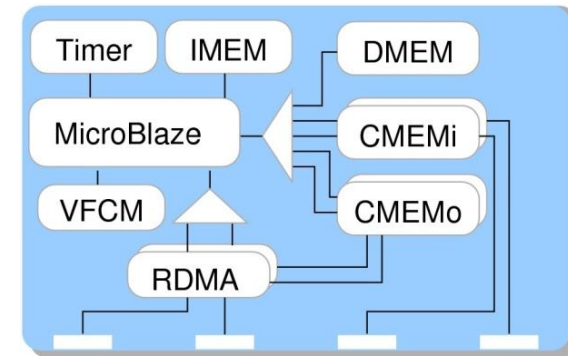
Predictable Execution

- ▶ We have access in bounded time to remote memories via the network
 - Next challenge is predictable execution on a shared processor tile
- ▶ **The ET of a task may depend on other tasks or resources**
 - Complicates analysis
 - **Independence from other resources**
 - Private instructions and data are stored in local memory
 - **Independence from other tasks**
 - Tasks not scheduled until tokens are available in CMEMi and there is sufficient empty space to store output tokens in CMEMo



Predictable Processor Scheduling

- ▶ **Scheduling in software is not done in a single cycle**
 - Time partitioned into **task slots** and **OS slots** of bounded length
 - Tasks execute in task slots, scheduler executes in OS slots
- ▶ **Non-real-time tasks may have unbounded execution times**
 - **Preemption** required for predictability
 - Timer interrupts at the end of a task slot
 - Atomizes task executions into tasks slots of bounded length
- ▶ **Processor is not arbitrarily preemptive**
 - E.g. reads to remote memories may take hundreds of cycles
 - All external communication must **use remote DMA**
 - Programmed via loads and stores in a single cycle
 - Polling for completion done in a single cycle
 - Preemption delay is hence very short



Predictable Processor Resource

- ▶ **Response times of real-time tasks are bounded**
 - Execution times of real-time tasks are bounded
 - Task slot durations are enforced by preemption
 - Tasks are scheduled by predictable arbiter
 - E.g. Static scheduling, TDM, or Credit-Controlled Static-Priority
 - Overhead of scheduling is bounded by OS slot

Presentation Outline

Introduction

CoMPSoC overview

Memory tile

Interconnect

Processing tile

Conclusions

Conclusions

- ▶ Real-time streaming systems get increasingly complex
 - Problem to verify that firm real-time requirements are satisfied
 - Simulation-based verification is circular and has poor coverage
- ▶ **Formal verification** is a promising alternative verification approach
 - Covers all possible interactions with the system
 - Relies on timing models of entire system (applications + platform)
 - Requires **predictable systems** for efficient implementation
- ▶ **CoMPSoC** is a predictable multi-processor platform
 - Predictable processor tiles with real-time operating system
 - Predictable network-on-chip
 - Predictable memory tiles supporting both SRAM and SDRAM
 - Supported by an automated design flow

Conclusions

- ▶ We presented **three general techniques** for predictable systems
 1. **Independent execution per resource** to simplify analysis
 - Require **all inputs** and **sufficient space** for output before scheduling
 2. **Predictable arbitration** enables bounded response times
 - Uses budgets (resource reservations) to regulate resource access
 - There are many known predictable arbiters to choose from
 3. **Preemption** protects against large or unbounded requests
 - **Atomization** of requests in memory controller and interconnect
 - Preemption in processor tile using interrupts

Questions?

k.b.akesson@tue.nl