Chapter 4: System Synthesis

Outline

- System design trends
  - Model-based synthesis
  - Transaction level model generation
  - Application to platform mapping
  - Platform generation
  - Cycle-accurate model generation
Traditional System Design

- **Hardware first approach**
  - Platform is defined by architect or based on legacy
  - Designers develop and verify RTL model of platform
  - Slow error prone process
- **SW development after HW is finalized**
  - Debugging is complicated on the board due to limited observability
  - HW errors found during SW development are difficult to rectify
- **Application is ported after system SW is finalized**

Virtual Platform based System Design

- **Virtual platform (VP) is a fast model of the HW platform**
  - Typically an instruction set simulator or C/C++ model of the processor
  - Peripherals are modeled as remotely callable functions
  - Executes several orders of magnitude faster than RTL
- **SW and HW development are concurrent**
  - VP serves as the golden model for both SW and HW development
  - SW development can start earlier
  - HW designers can use SW for realistic test bench for RTL
Model-based System Design

- **Model based design gives control to application developers**
  - Application is captured as high level C/C++/UML specification
  - Transaction level model (TLM) is used to verify and evaluate the design

- **System synthesis**
  - The best platform for given application can be synthesized automatically
  - For legacy platforms, application mapping can be generated automatically
  - Cycle accurate SW/HW can be generated from TLM for implementation

Outline

- **System design trends**

  → **Model-based synthesis**

  - Transaction level model generation
  - Application to platform mapping
  - Platform generation
  - Cycle-accurate model generation
Model Based Synthesis

- Synthesis of cycle-accurate model (CAM) from specification
  - Process may be divided into several steps
  - Specification is defined as application model and design constraints
  - Several intermediate models, such as TLMs, may be used
  - Platform component models are needed for TLM generation

System Synthesis Inputs and Output

- Inputs
  - Application Model
    - Purely functional model
    - Specified in a given model of computation (Stateflow, dataflow, CSP, MP)
  - Component Models
    - Data models of configurability and metrics
    - Functional models of component services
    - Examples: HW IP models (Processor, Peripheral, Bus), SW IP models (RTOS, Drivers)
  - Constraints
    - Bounds on metrics (Performance, area, power, reliability, security)
    - Optimization goal as a cost function of metrics

- Output
  - TLM of application mapped to HW/SW platform
Three Models with Respect to OSI (Ref. Chapter 3)

Cycle Accurate Model
Transaction Level Model
Specification Model

TLMs

Address lines
Data lines
Control lines

Outline

- System design trends
- Model-based synthesis
  - Transaction level model generation
    - Application to platform mapping
    - Platform generation
    - Cycle-accurate model generation
Synthesis Case 1: Fixed Platform and Mapping

- Initial platform and mapping are given
- Optimization tools may modify spec under given constraints

Tool support for Synthesis Case 1

- GUI for application specification
- GUI for platform specification
- GUI for application to platform mapping
- TLM generation tool
- TLM-based metric estimation tools
- Constraint-based spec optimization tools
Input: Application Model

- Application model consists of
  - Processes for computation (e.g., P1, P2, P3, P4)
  - Channels for communication (e.g., C1 between P1 and P3)
  - Variables for storage (e.g., v1)

Application Model Objects

- Processes
  - Symbolic representation of computation
  - Contain C/C++ code imported from reference

- Process ports
  - Symbolic representation of communication services required by processes
  - Provide object orientation by allowing processes to connect to different channels

- Channels
  - Symbolic representation of inter-process communication
  - Implement communication services such as blocking, non-blocking, handshake, FIFO etc.
  - Encapsulation for communication functions

- Variables
  - Symbolic representation of data storage
Input: Platform Architecture

- Platform consists of
  - Hardware: PEs (e.g., CPU1, HW), Buses (e.g., Bus1), Memories (e.g., Mem), Interfaces (e.g., Transducer)
  - Software: Operating systems (e.g., OS1) on SW PEs

Platform Objects

- Processing element (PE)
  - Symbolic representation of computation resources
  - Different types such as SW processors, HW IPs etc.

- Bus
  - Symbolic representation of communication media
  - Types include shared, point-to-point, link, crossbar etc.

- Memory
  - Symbolic representation of physical storage
  - May contain shared variables or SW program/data

- Transducer
  - For protocol conversion and store-forward routing
  - Necessary for PEs with different bus protocols

- Operating system (OS)
  - Software platform for individual PEs
  - Needed for scheduling multiple processes on a PE
**Input: Mapping**

![Diagram showing mapping of processes, channels, and variables](image)

**Processes → PEs**
- Each process in the application must be mapped to a PE
- Multiple processes may be mapped to SW PE with OS support
- Example: P1, P2 → CPU1

**Channels → Routes**
- All channels between processes mapped to different PEs are mapped to routes in the platform
- Route consists of bus segments and interfaces
- Channel on each bus segment is assigned a unique address

**Variables → Memories**
- Variables accessed by processes mapped to different PEs are mapped to shared memories
- All variables are assigned an address range depending on size

**Mapping Rules**
**Computation Timing Estimation**

- Stochastic memory delay model
- DFG scheduling to compute basic block delay [DATE 08]
- RTOS model added for PEs with multiple processes

**Stochastic Memory Delay Model**

- **Assumption**
  - Cache and branch prediction hit rate available in data model
- **Delay Estimation**
  - Operation access overhead = \( N_{\text{op}} \times ((1.0 - \text{HR}) \times (\text{CD} + L_{\text{mem}})) \)
  - Data access overhead = \( N_{\text{id}} \times ((1.0 - \text{HR}) \times (\text{CD} + L_{\text{mem}})) \)
  - Branch prediction miss penalty = \( \text{MP}_r \times \text{Penalty} \)
**Processor Timing Estimation**

- **Assumptions**
  - In-order, single issue processor
  - Optimistic during scheduling (100% cache hit)

Operations |
---|---|
Add | Int-ALU Qty: 1 IntAdd IntSub Lat: 1 Lat: 1
Sub

Datapath

Processor Data Model

1: \(a = i - 1\)
2: \(t_1 = a + 2\)
3: \(d_2 = n \times m\)
4: \(t_3 = t_1 - t_2\)
5: load \(b\)
6: \(t_4 = b / 10\)
7: jmp
8: wait \(47 \times CT\)

Total BB delay = Op.+Mem.+Br. = 47.3 cycles

Operation delay = 42

**Communication Timing Estimation**

- Protocol model used to estimate synchronization, arbitration and transfer
- Timing is annotated in bus channel

Write() {
    Get_Bus();
    Transfer();
    Release_Bus();
}

Write() {
    Get_Bus();
    wait (t1);
    Transfer();
    wait (t2);
    Release_Bus();
    wait (t3);
}
**Output: SystemC Timed TLM**

**TLM Generation Technique**
- Application code \(\rightarrow\) sc_thread
- Processing element \(\rightarrow\) sc_module
- OS Model \(\rightarrow\) sc_module
- Bus \(\rightarrow\) sc_channel
- Memory \(\rightarrow\) Array inside sc_module
- Interface \(\rightarrow\) FIFO channel+sc_process

**Outline**

- System design trends
- Model-based synthesis
- Transaction level model generation
  - Application to platform mapping
  - Platform generation
  - Cycle-accurate model generation
Application to Platform Mapping

- Mapping is derived from Application and Platform
- Optimization loop is driven by estimation results and constraints

Application Example

- GSM Encoder
  - Compresses raw speech data frame-by-frame
  - Over 10K lines of C code in specification
  - 5 top level functions: LP, OP, CL, CB, UP
  - Contains if-then-else and loop control flow
Profiling

- **Given input MoC, profile application for:**
  - **Computation**
    - Number of operations (size)
    - Operations type per data type and frequency of use
    - Concurrency between modules and dependency
  - **Communication**
    - Volume, frequency of communication between modules
    - Timing dependency
    - Latency requirements
  - **Storage**
    - Instruction size
    - Variable size

**Profiling helps select the appropriate components for implementation**
- All fixed point ops ➔ No need for processors with floating point units
- Large number of multiplications ➔ Processor with HW multiplier is ideal
- CB is most computationally intensive ➔ Ideal for custom HW mapping

**Profiling Statistics**

- **Encoder:** 8,802
- **LP Analysis:** 377 MOp
- **Closed Loop:** 479.7 MOp
- **Codebook:** 646.5 MOp
- **Update:** 43.6 MOp

- Others:
  - (+, int), 46.20%
  - (*, int), 33.50%
  - (+, Int), 4.10%
  - (*, Int), 9.10%
**Application Graph**

- Profile information is abstracted into a simplified graphical representation for synthesis algorithms
  - Node tags = number of operations in the process (#ops) in millions
  - Edge tags = kilobytes transferred between the processes
  - Control dependencies are excluded for simplicity

**Platform Connectivity Graph**

- Platform architecture is abstracted into a connectivity graph showing possibility of inter-PE communication
  - Node label = PE name
  - Node tag = estimated computation speed of the PE in Million of Operations per second (Mops)
  - Edge implies a communication path between PEs
    - No edge between HW IP and DSP due to missing DMA on Bus1
Load Balancing (LB) Algorithm

- Greedy heuristic to map processes to least busy PEs

- PE load = total time PE will be busy executing the mapped processes
  - Defined as $\sum \#\text{ops}(p) / \text{Speed (PE)}$, for all $p$ mapped to $PE$
  - Does not account for any communication time!

- Feasibility list defined for each process
  - The set of PEs to which a process can be mapped such that the process’ communication requirements are not violated
    - Let processes $p$ & $q$ are have an edge in the application graph
    - Let $q$ be already mapped to $PE'$
    - $PE$ is in feasible($p$) if there exists edge ($PE, PE'$) in platform graph

- Basic idea
  - Map processes to least loaded feasible PE in decreasing order of $\#\text{ops}$

LB Algorithm in Action

(a) Application graph

(b) Platform connectivity graph w/ mapping
Longest Processing Time (LPT) Algorithm

- **Drawbacks of LB**
  - Does not account for communication
  - May not terminate with a mapping if feasibility list is empty
  - LPT accounts for communication and always produces a mapping

- **Fully connected platform**
  - DMA is added to allow HW peripherals to communicate with processes and memories
  - No need to evaluate feasibility of mapping

- **Cost of mapping process to PE is defined**
  - Includes both communication and computation time

- **Basic idea**
  - Map processes to least cost PE in decreasing order of #ops

LPT Cost Function

- \( E(p, PE) \): Estimated time for running \( p \) on \( PE \)
  - \( \text{Mops}(p) / \text{Speed}(PE) \) + time to send data to mapped processes

- \( C(p, PE) = T(PE) + E(p, PE) - \text{SystemEndTime} \)

- PE with lowest execution time may not have the lowest cost
  - \( E(p, PE3) > E(p, PE2) \), but \( C(p, PE3) < C(p, PE2) \)
New Connectivity Graph of Updated Platform

- Platform architecture is abstracted into a connectivity graph
  - Node label = PE name, Node tag = estimated PE speed in Mops
  - Edge = Connectivity between PEs
  - Edge label = effective transaction speed between PEs in Kilobytes per second (Kbps): added for LPT algorithm

LPT Algorithm in Action

(a) Application graph

(b) Platform connectivity graph w/ mapping
Outline

• System design trends
• Model-based synthesis
• Transaction level model generation
• Application to platform mapping

Platform generation

• Cycle-accurate model generation

Platform Generation

[Diagram showing the process of platform generation with steps from System Level Specification, Application, Constraints, Component Models, Platform Generation, Mapping, Platform, TLM Generation, Optimization, Estimation, Metrics.]
## Component Database

<table>
<thead>
<tr>
<th>PE Type</th>
<th>Cost</th>
<th>Speed</th>
<th>Capacity (Speed * 6 sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>2</td>
<td>100</td>
<td>600</td>
</tr>
<tr>
<td>DSP</td>
<td>1</td>
<td>50</td>
<td>300</td>
</tr>
<tr>
<td>HW</td>
<td>5</td>
<td>200</td>
<td>1200</td>
</tr>
</tbody>
</table>

- **Timing constraint:** Application must complete in <6 seconds.
- **Database of processing elements used for component selection**
  - Characterized by type, cost and speed
  - Cost includes IP licensing, development, manufacturing etc.
  - PE Computation \( \text{Capacity} = \text{PE speed (in Mops)} \times \text{timing constraint} \)
  - Indicates number of operations (in millions) that may be mapped to a PE while still meeting the timing constraints

## Platform Generation Algorithm

- **Greedy heuristic to minimize cost and meet timing constraint**
- **PE slack**
  - Capacity remaining on the PE
  - \( \text{Slack(PE)} = \text{Capacity(PE)} - (\sum \#\text{ops}(p), \text{for all } p \text{ mapped to PE}) \)
- **Closeness factor of a process \( p \) to a PE**
  - Total communication data between \( p \) and all processes mapped to PE
  - \( C(p, PE) = \sum (\text{Edge-weight}(p, q) \text{ in app. graph}), \text{for all } q \text{ mapped to PE} \)
- **Basic idea**
  - Iterate over processes \( (p) \) with in decreasing order of \#ops
    - Update slacks of allocated PEs
    - Map \( p \) to closest PE with \( \text{Slack(PE)} \geq \#\text{ops}(p) \)
    - Allocate least cost PE with \( \text{Capacity} \geq \#\text{ops}(p) \) if mapping fails
Platform Generation Algorithm in Action

(a) Application graph
(b) Generated Platform w/ mapping

Outline

- System design trends
- Model-based synthesis
- Transaction level model generation
- Application to platform mapping
- Platform generation

→ Cycle-accurate model generation
CAM Generation

SystemC TLM

SW/RTOS Library

RTL IP Library

SW Synthesis

C→RTL

Interface Synthesis

Binary

HW RTL

IF RTL

Pin/Cycle Accurate Model (PCAM)

Generator

CA Sim. Tools

C/Verilog CAM

FPGA Tools

Bus Library

Prototype

Cycle-Accurate Software Synthesis (Chapter 5)

- Processes → Compiled App.
- OS model → Real OS
- HAL model → Real HAL
Cycle-Accurate Hardware Synthesis (Chapter 6)

- Process → Synthesizable RTL
- High level synthesis for custom
- Replacement for HW IP

Cycle-accurate Synthesis

CPU1

Mem

Bus1

TX

Bus2

P3

HW IP (RTL)

Processes in C

CPU2

Cycle-Accurate Interface Synthesis (Chapter 7)

- Sync. Model → Interrupts
- Bus channel → Arbiter + Signals
- Interface model → RTL
- Channel access → PE interface

CPU1

Mem

Arbiter

IC

TX

HW IP

TX

CPU2

Interface Synthesis
Cycle-Accurate Model

PCAM is downloaded automatically for fast prototyping with FPGAs or simulated using validation tools.

Summary

- **Emergence of model-based system design**
  - Virtual platforms replace prototypes for early SW development
  - Increasing adoption of TLMs for SW/HW design
- **Challenges for synthesis of large system designs**
  - Manual model development is time consuming and error-prone
  - Different platforms are needed for different application domains
  - Mapping application to a multi-core platform is complicated
- **Need for well defined model semantics is needed at TLM and cycle-accurate levels**
  - Enables automatic TLM generation
  - System synthesis becomes possible
- **Future of system synthesis**
  - Based on formalized system level models such as TLM
  - Automatic mapping of application to platform
  - Automatic generation of application specific platforms