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 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)

<table>
<thead>
<tr>
<th>Cycle Accurate Model</th>
<th>Transaction Level Model</th>
<th>Specification Model</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Spec</td>
</tr>
<tr>
<td>7. Application</td>
<td></td>
<td>7. Application</td>
</tr>
<tr>
<td>6. Presentation</td>
<td></td>
<td>6. Presentation</td>
</tr>
<tr>
<td>5. Session</td>
<td></td>
<td>5. Session</td>
</tr>
<tr>
<td>4. Transport</td>
<td></td>
<td>4. Transport</td>
</tr>
<tr>
<td>2b. Link + Stream</td>
<td></td>
<td>2b. Link + Stream</td>
</tr>
<tr>
<td>2a. Media Access Ctrl</td>
<td></td>
<td>2a. Media Access Ctrl</td>
</tr>
<tr>
<td>1. Physical</td>
<td></td>
<td>1. Physical</td>
</tr>
</tbody>
</table>

#### 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 (eg. P1, P2, P3, P4)
  - Channels for communication (eg. C1 between P1 and P3)
  - Variables for storage (eg. 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

- Processes $\rightarrow$ PEs
- Channels $\rightarrow$ Routes
- Variables $\rightarrow$ Memories

**Mapping Rules**

- **Processes to 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 $\rightarrow$ CPU1

- **Channels to 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 to Memories**
  - Variables accessed by processes mapped to different PEs are mapped to shared memories
  - All variables are assigned an address range depending on size
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 = \( N_{op} \times ((1.0 - HR_i) \times (CD + L_{mem})) \)
  - Data access overhead = \( N_{ld} \times ((1.0 - HR_d) \times (CD + L_{mem})) \)
  - Branch prediction miss penalty = \( MP_{rate} \times \text{Penalty} \)

```
1: a = $i - 1
2: t1 = a + 2
3: t2 = $n \times $m
4: t3 = t1 - t2
5: load b
6: t4 = b / 10
7: jmp
```

**Memory/Br. Delay Calculation**

Mem. Overhead = 4.1
Branch Delay = 1.2
Processor Timing Estimation

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

  ```
  1: a = i - 1  
  2: t1 = a + 2  
  3: t2 = n * m  
  4: t3 = t1 - t2  
  5: load b  
  6: t4 = b / 10  
  7: jmp  
  8: wait 47*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( );  
  }
  ```
Output: SystemC Timed TLM

TLM Generation Technique

- Application code → sc_thread
- Processing element → sc_module
- OS Model → sc_module
- Bus → sc_channel
- Memory → Array inside sc_module
- Interface → 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

Profiled Statistics

- 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
Application Graph

- Profile information is abstracted into a simplified graphical representation for synthesis algorithms
  - Node tags = millions of operations
  - Edge tags = kilobytes transferred
  - Control dependencies are excluded for simplicity

Platform Connectivity Graph

- Platform architecture is abstracted into a connectivity graph showing possibility of inter-PE communication
  - Node tag = PE speed (relative)
  - No edge between HW and DSP due to missing DMA on Bus1
Load Balancing Algorithm

(a) Application graph

(b) Platform connectivity graph w/ mapping

Connectivity Graph of Updated Platform

- Platform architecture is abstracted into a connectivity graph
  - Node = PE, Node label = Relative PE speed
  - Edge = Path between PEs, Edge label = Relative communication delay
### Longest Processing Time Algorithm

![Diagram showing the algorithm steps]

1. **Start**
2. Map f with max. @Ops to fastest PE
3. Select unmapped f with max. @Ops
4. Map f to PE with min. Cost (PE)
5. Update completion times and slack
6. Are all functions mapped?
   - Yes: **Done**
   - No: go back to step 2

#### LPT Cost Function Computation

Cost is computed as the timing overhead of selecting a PE
- System end time and PE costs are updated at each LPT step
- PE with lowest execution time may not have the lowest cost
- LPT terminates when all functions are mapped
Longest Processing Time Algorithm Result

(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

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
  - Computation capacity is the PE speed multiplied by timing constraint
  - Similar library for buses and memories may be used
Platform Generation Algorithm

Outline

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

![Diagram of CAM Generation]

**Cycle-Accurate Software Synthesis (Chapter 5)**

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

- Process $\rightarrow$ Synthesizable RTL
- High level synthesis for custom
- Replacement for HW IP

Cycle-Accurate Interface Synthesis (Chapter 7)

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