Embedded System Design

Modeling, Synthesis, Verification

Daniel D. Gajski, Samar Abdi, Andreas Gerstlauer, Gunar Schirner

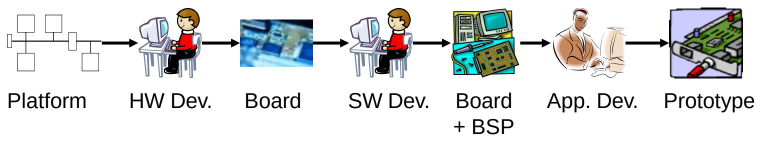
Chapter 4: System Synthesis

9/29/2011

Outline

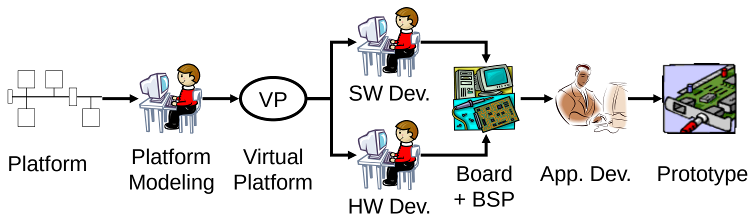
➡ System design trends
- Model-based synthesis
- Transaction level model generation
- Application to platform mapping
- Platform generation
- Cycle-accurate model generation

Traditional System Design



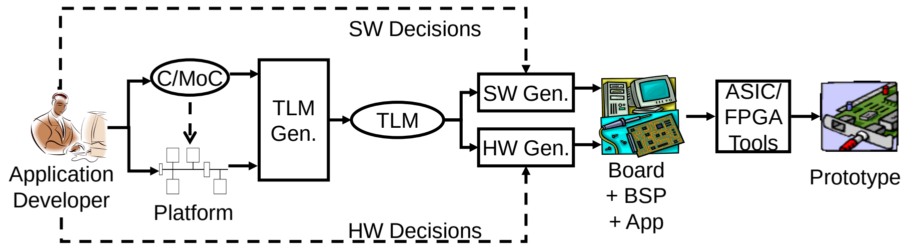
- **Hardware first approach**
  - Platform is defined by architect or based on legacy
  - Designers develop and verify RTL model of platform
  - Slow error prone process
- **SW development after HW is finalized**
  - Debugging is complicated on the board due to limited observablity
  - HW errors found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

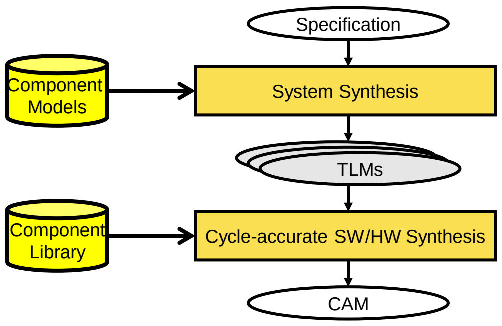


- **Model based design gives control to application developers**
  - Application is captured as high level C/C++/UML specification
  - Transaction level model (TLM) is used to verify and evaluate the design
- **System synthesis**
  - The best platform for given application can be synthesized automatically
  - 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

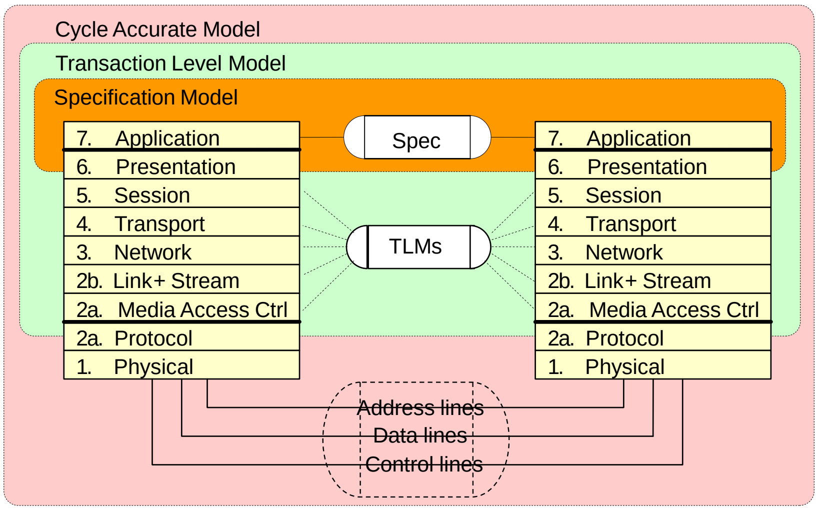


- **Synthesis of cycle-accurate model (CAM) from specification**
  - **Process may be divided into several steps**
  - **Specification is defined as application model and design constraints**
  - **Several intermediate models, such as TLMs, may be used**
  - **Platform component models are needed for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

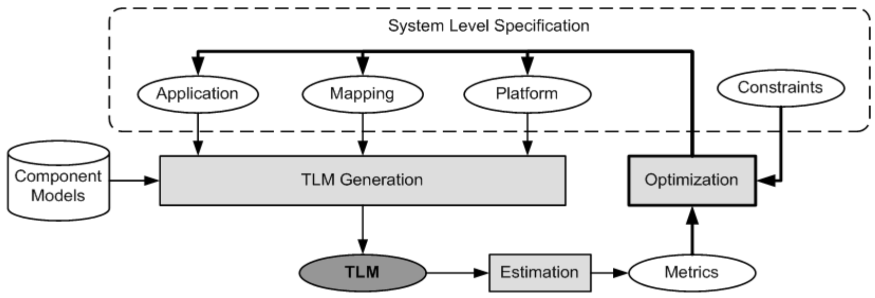
Three Models with Respect to OSI (Ref. Chapter 3)



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

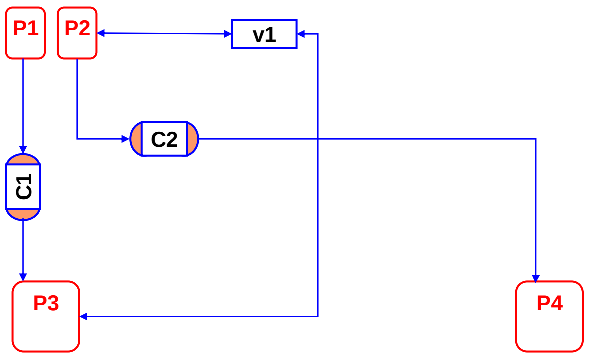


- **Initial platform and mapping are given**
- **Optimization tools may modify spec under given constraints**

Tool support for Synthesis Case 1

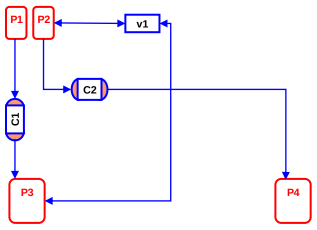
- GUI for application specification
- GUI for platform specification
- GUI for application to platform mapping
- TLM generation tool
- TLM-based metric estimation tools
- Constraint-based spec optimization tools

Input: Application Model



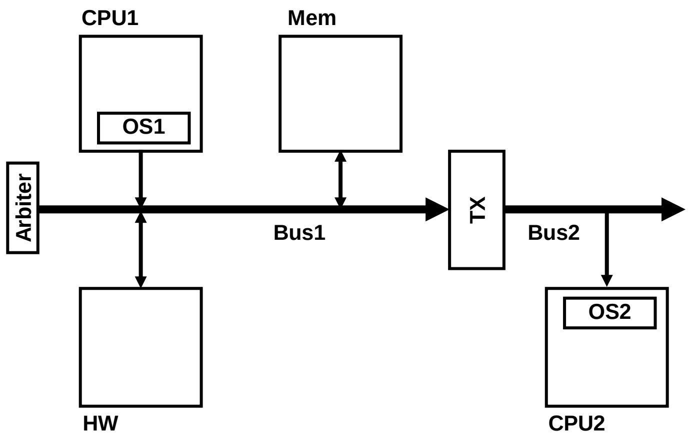
- **Application model consists of**
  - Processes for computation (eg. P1, P2, P3, P4)
  - Channels for communication (eg. C1 between P1 and P3)
  - Variables for storage (eg. v1)

Application Model Objects



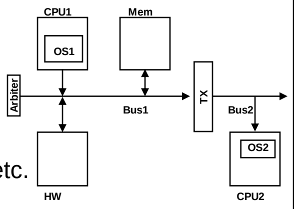
- **Processes**
  - Symbolic representation of computation
  - Contain C/C++ code imported from reference
- **Process ports**
  - Symbolic representation of communication services required by processes
  - Provide object orientation by allowing processes to connect to different channels
- **Channels**
  - 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



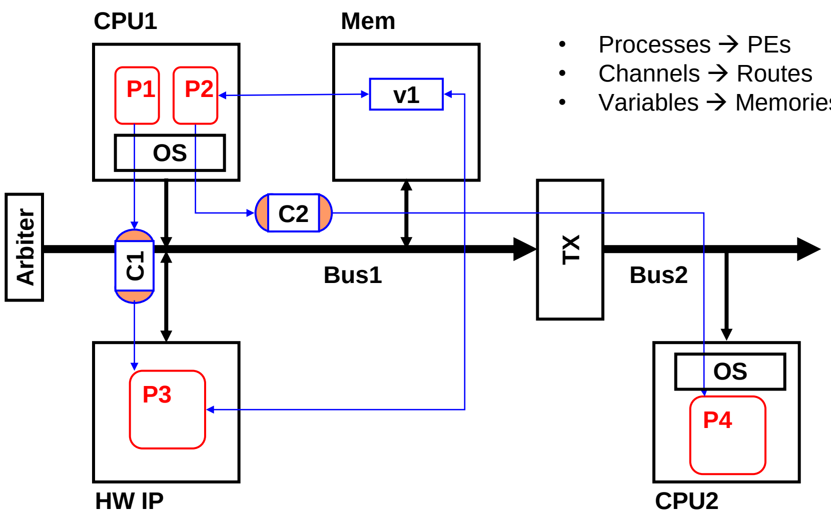
- **Platform consists of**
  - Hardware: PEs (eg. CPU1, HW), Buses (eg. Bus1), Memories (eg. Mem), Interfaces (eg. Transducer)
  - Software: Operating systems (eg. OS1) on SW PEs

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

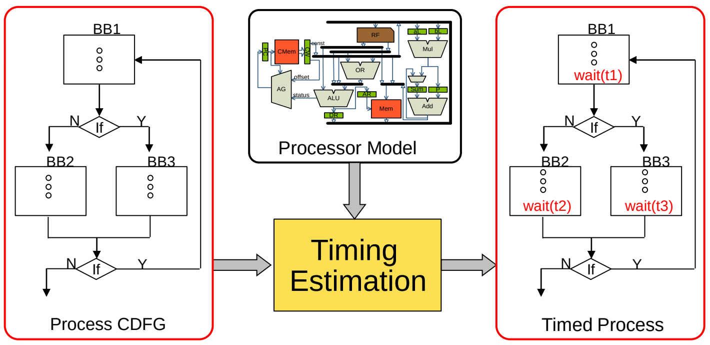


- Processes → PEs
- Channels → Routes
- Variables → Memories

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 → 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

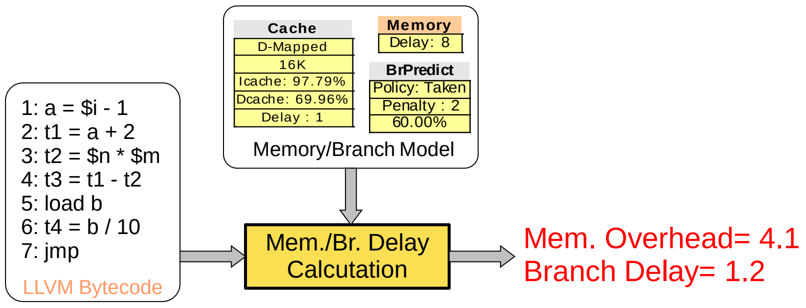
Computation Timing Estimation



- 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} * ((1.0 – HR_i) * (CD + L_{mem}))
  - Data access overhead = N_{ld} * ((1.0 – HR_d) * (CD + L_{mem}))
  - Branch prediction miss penalty = MP_{rate} * Penalty

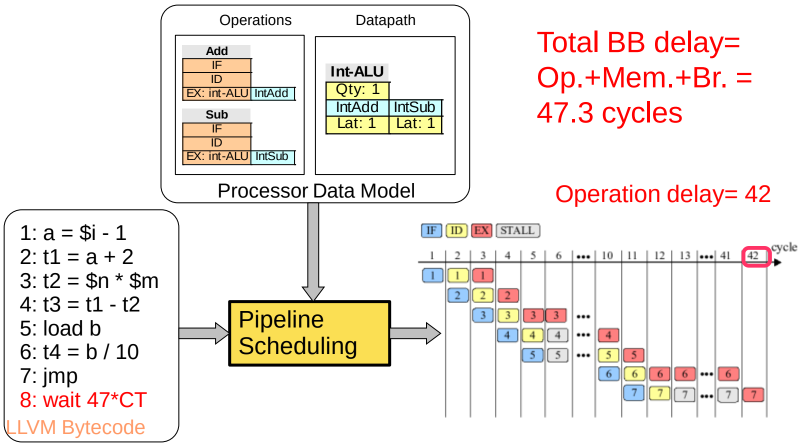


```
1: a = $i - 1
2: t1 = a + 2
3: t2 = $n * $m
4: t3 = t1 - t2
5: load b
6: t4 = b / 10
7: jmp
```

Mem. Overhead= 4.1
Branch Delay= 1.2

Processor Timing Estimation

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

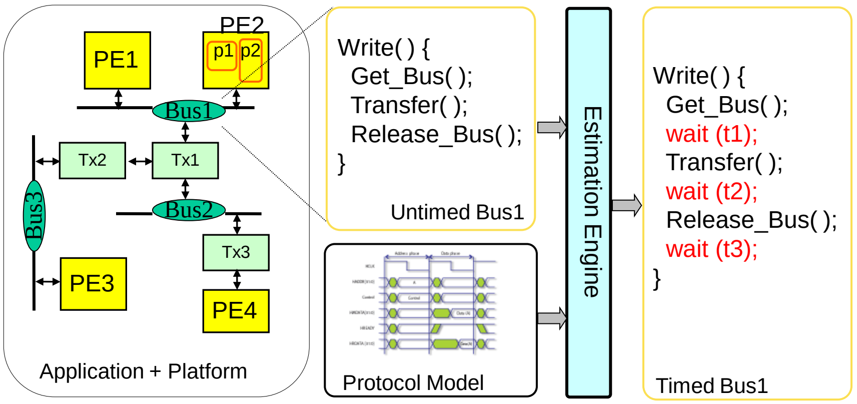


```
1: a = $i - 1
2: t1 = a + 2
3: t2 = $n * $m
4: t3 = t1 - t2
5: load b
6: t4 = b / 10
7: jmp
8: wait 47*CT
```
LLVM Bytecode

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

Operation delay= 42

Communication Timing Estimation

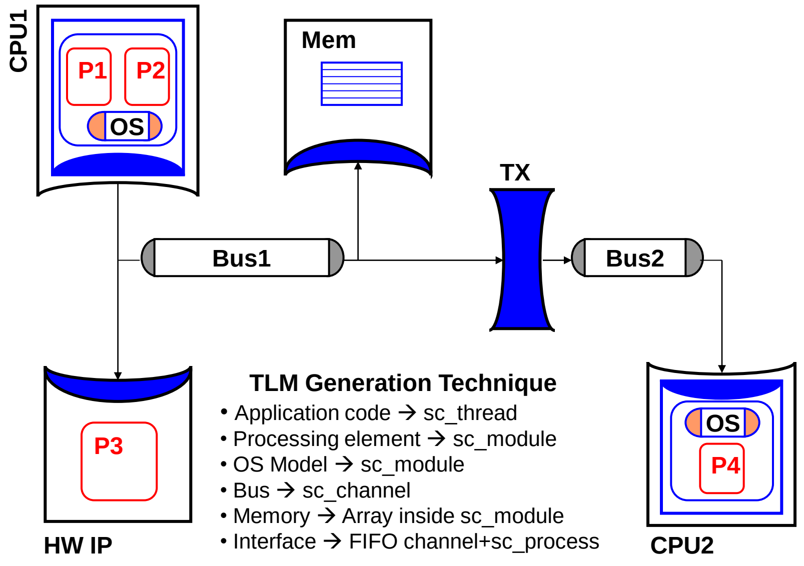


```
Write( ) {
  Get_Bus( );
  Transfer( );
  Release_Bus( );
}
```
Untimed Bus1

```
Write( ) {
  Get_Bus( );
  wait (t1);
  Transfer( );
  wait (t2);
  Release_Bus( );
  wait (t3);
}
```
Timed Bus1

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

Output: SystemC Timed TLM



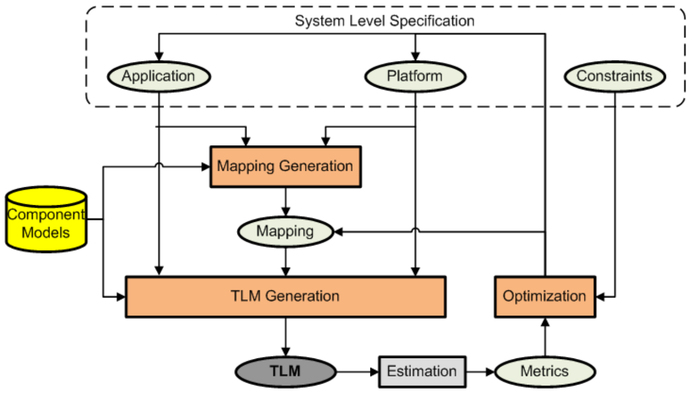
TLM Generation Technique

- 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

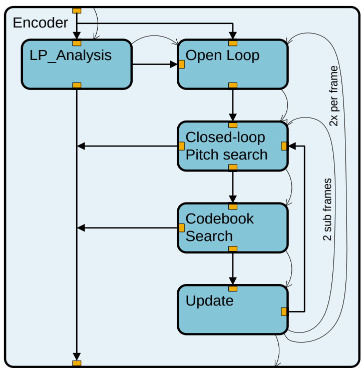
- **System design trends**
- **Model-based synthesis**
- **Transaction level model generation**
- ➡ **Application to platform mapping**
- **Platform generation**
- **Cycle-accurate model generation**

Application to Platform Mapping



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

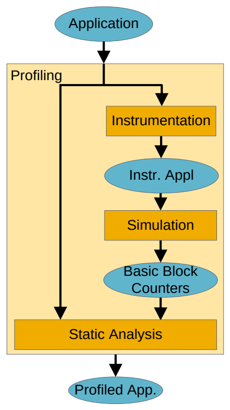
Application Example



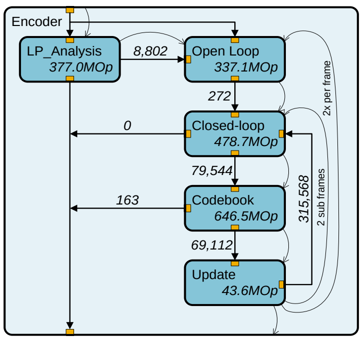
- **GSM Encoder**
  - Compresses raw speech data frame-by-frame
  - Over 10K lines of C code in specification
  - 5 top level functions: LP, OP, CL, CB, UP
  - Contains if-then-else and loop control flow

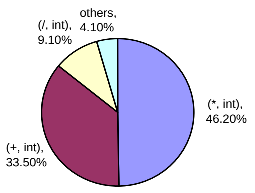
Profiling

- **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



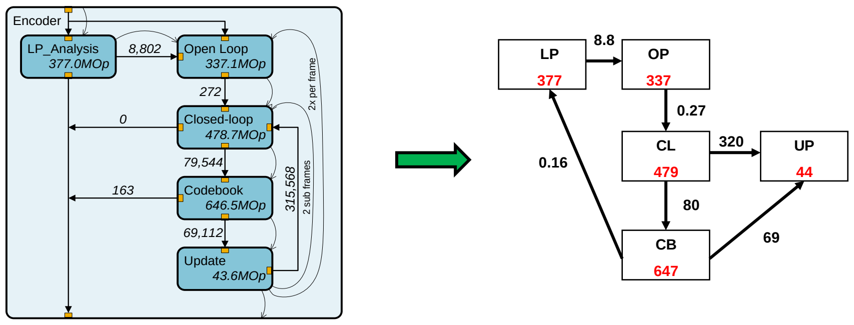
Profiled Statistics





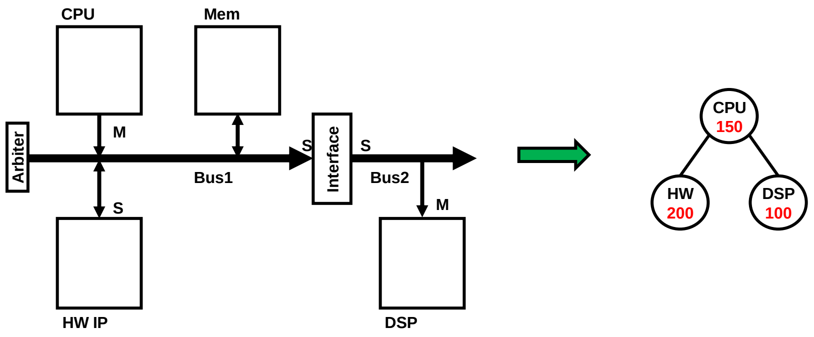
- **Profiling helps select the appropriate components for implementation**
  - All fixed point ops➔ No need for processors with floating point units
  - Large number of multiplications ➔ Processor with HW multiplier is ideal
  - CB is most computationally intensive ➔ Ideal for custom HW mapping

Application Graph



- **Profile information is abstracted into a simplified graphical representation for synthesis algorithms**
  - Node tags = 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

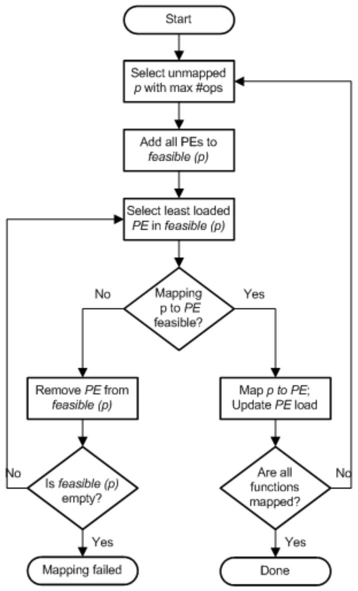


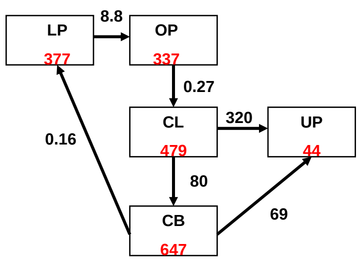
- **Platform architecture is abstracted into a connectivity graph showing possibility of inter-PE communication**
  - Node label = PE name
  - Node tag = estimated computation speed of the PE in Million of Operations per second (Mops)
  - Edge implies a communication path between PEs
    - No edge between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

Load Balancing (LB) Algorithm

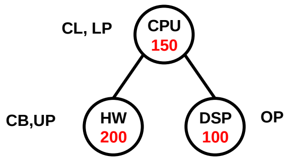
- **Greedy heuristic to map processes to *least busy* PEs**
- **PE load = total time PE will be busy executing the mapped processes**
  - Defined as ∑*#ops(p) / Speed (PE),* for all *p* mapped to *PE*
  - Does not account for any communication time!
- **Feasibility list defined for each process**
  - 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 #ops

LB Algorithm in Action


Start
Select unmapped p with max #ops
Add all PEs to feasible (p)
Select least loaded PE in feasible (p)
Mapping p to PE feasible?
No
Yes
Remove PE from feasible (p)
Map p to PE; Update PE load
Is feasible (p) empty?
No
Yes
Mapping failed
Are all functions mapped?
No
Yes
Done



(a) Application graph

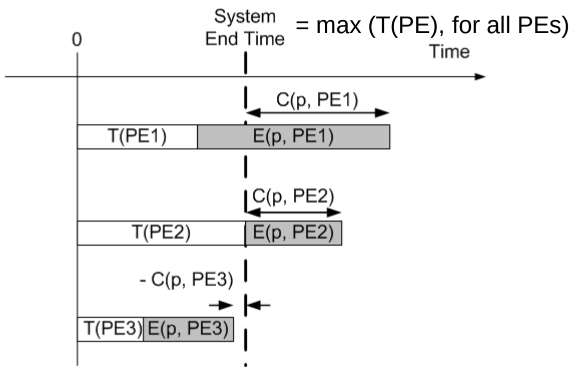


(b) Platform connectivity graph w/ *mapping*

Longest Processing Time (LPT) Algorithm

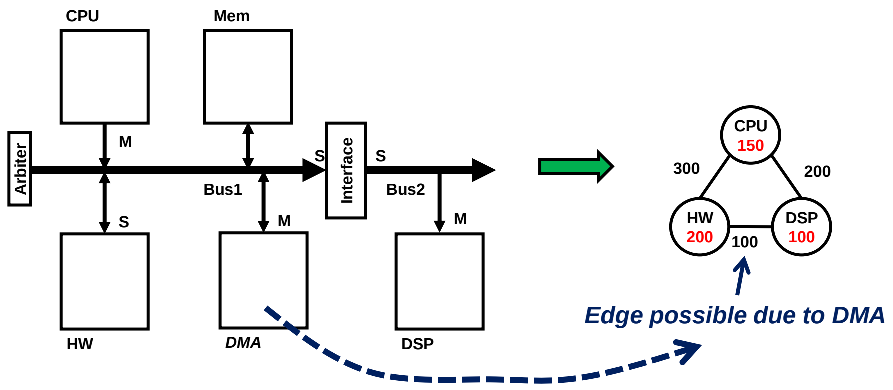
- **Drawbacks of LB**
  - Does not account for communication
  - May not terminate with a mapping if feasibility list is empty
  - LPT accounts for communication and always produces a mapping
- **Fully connected platform**
  - DMA is added to allow HW peripherals to communicate with processes and memories
  - 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



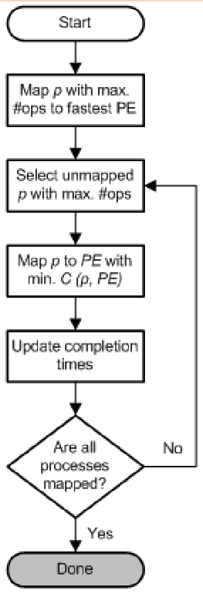
- ***E(p, PE):* Estimated time for running *p* on *PE***
  - *Mops(p) / Speed (PE)* + time to send data to mapped processes
- ***C(p, PE) = T(PE) + E(p, PE) – SystemEndTime***
- **PE with lowest execution time may not have the lowest cost**
  - E(p, PE3) > E(p, PE2), but C(p, PE3) < C(p, PE2)

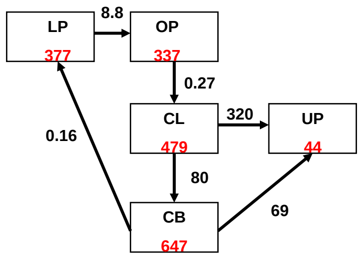
New Connectivity Graph of Updated Platform



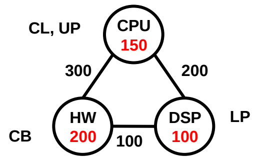
- **Platform architecture is abstracted into a connectivity graph**
  - Node label= PE name, Node tag= estimated PE speed in Mops
  - Edge = Connectivity between PEs
  - Edge label = effective transaction speed between PEs in Kilobytes per second (Kbps): *added for LPT algorithm*

LPT Algorithm in Action


Start
Map p with max. #ops to fastest PE
Select unmapped p with max. #ops
Map p to PE with min. C (p, PE)
Update completion times
Are all processes mapped?
No
Yes
Done



(a) Application graph

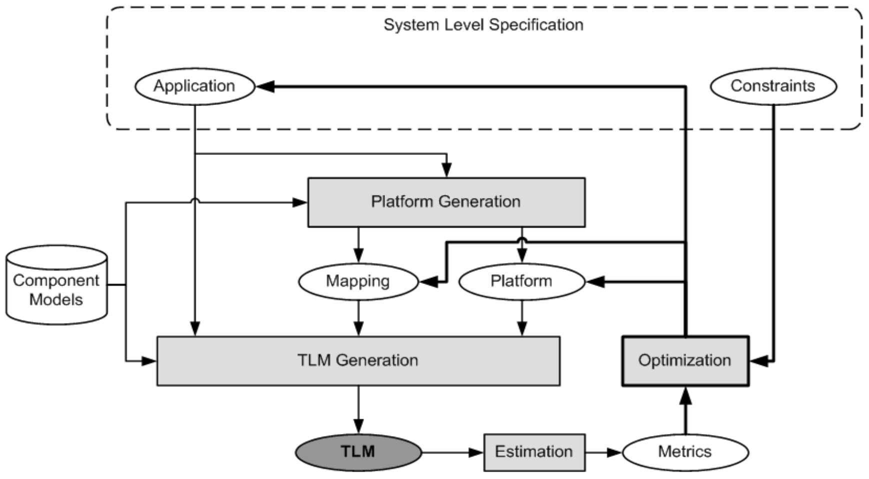


(b) Platform connectivity graph w/ mapping

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

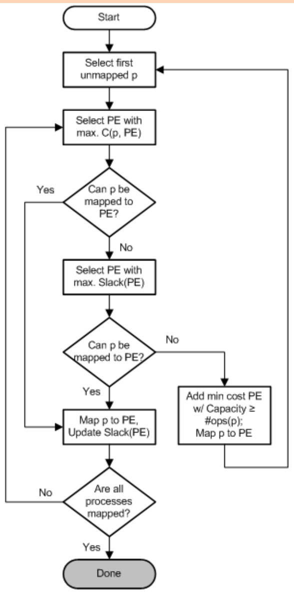
PE Type	Cost	Speed	Capacity (Speed *6 sec)
CPU	2	100	600
DSP	1	50	300
HW	5	200	1200

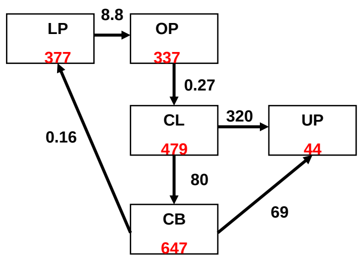
- **Timing constraint: Application must complete in <6 seconds.**
- **Database of processing elements used for component selection**
  - Characterized by type, cost and speed
    - Cost includes IP licensing, development, manufacturing etc.
  - PE Computation *Capacity* = PE speed (in Mops) * timing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

Platform Generation Algorithm

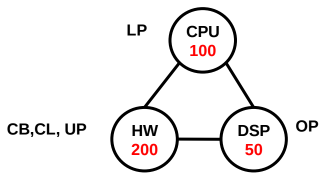
- **Greedy heuristic to minimize cost and meet timing constraint**
- **PE slack**
  - Capacity remaining on the PE
  - *Slack(PE) = Capacity(PE) – (∑#ops(p), for all p mapped to PE)*
- **Closeness factor of a process p to a PE**
  - Total communication data between *p* and all processes mapped to *PE*
  - *C(p, PE) = ∑ (Edge-weight(p, q) in app. graph), for all q 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 Slack(PE) ≥ #ops(*p*)
    - Allocate least cost PE with Capacity ≥ #ops(p) if mapping fails

Platform Generation Algorithm in Action


Start
Select first unmapped p
Select PE with max. C(p, PE)
Can p be mapped to PE?
Yes
No
Select PE with max. Slack(PE)
Can p be mapped to PE?
No
Yes
Add min cost PE w/ Capacity ≥ #ops(p); Map p to PE
Map p to PE, Update Slack(PE)
Are all processes mapped?
No
Yes
Done



(a) Application graph

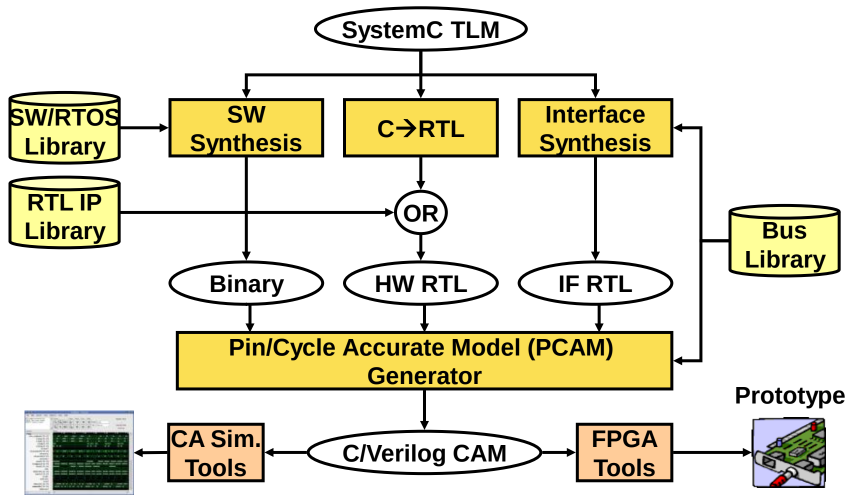


(b) Generated Platform w/ mapping

Outline

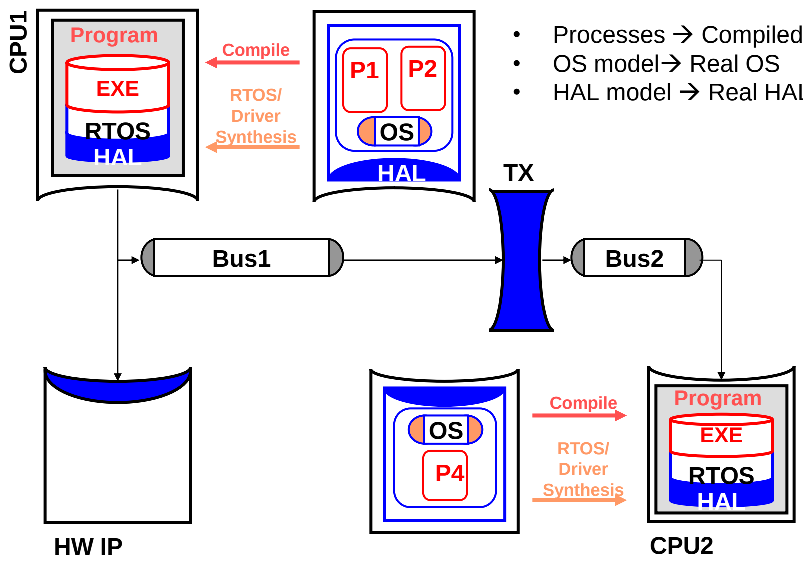
- System design trends
- Model-based synthesis
- Transaction level model generation
- Application to platform mapping
- Platform generation
- ➡ Cycle-accurate model generation

CAM Generation

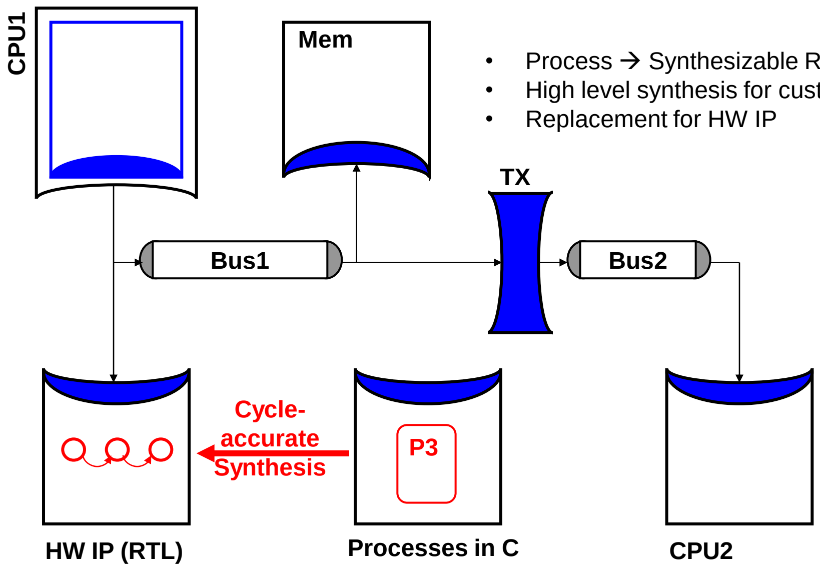


Cycle-Accurate Software Synthesis (Chapter 5)

- Processes → Compiled App.
- OS model→ Real OS
- HAL model → Real HAL

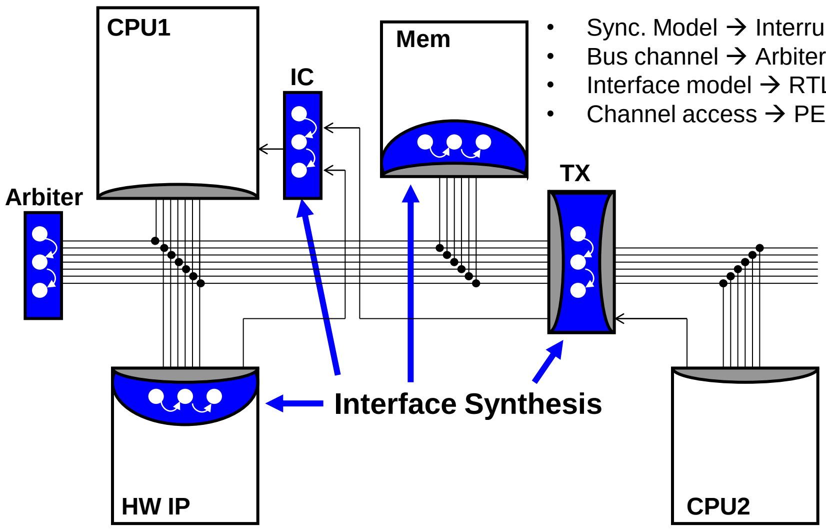


Cycle-Accurate Hardware Synthesis (Chapter 6)



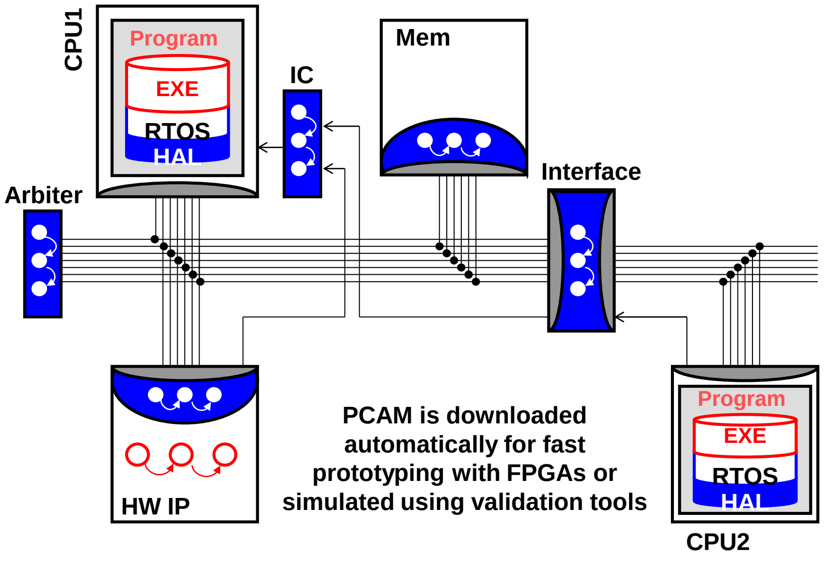
- Process → Synthesizable RTL
- High level synthesis for custom
- Replacement for HW IP

Cycle-Accurate Interface Synthesis (Chapter 7)



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

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