EEL 5722C
Field-Programmable Gate Array Design

Lecture 22: HW/SW Codesign:
Industry Practice and Academic Research*

Prof. Mingjie Lin

* Copyright © 1995-1999 SCRA
Rapid Prototyping Design Process

REUSE DESIGN LIBRARIES AND DATABASE

Primarily software

VIRTUAL PROTOTYPE

Primarily hardware

SYSTEM DEF.

FUNCTION DESIGN

HW & SW PART.

HW DESIGN

SW DESIGN

HW FAB

INTEG. & TEST

HW & SW CODESIGN

Partitioning & Codesign
Sanders Codesign Methodology

[HOOD94]
Sanders Codesign Methodology
Integrated Modeling Substrate

System Requirements
Arch Ind.
Proc Model

Hardware
Perf. Model
Behavior
Level Model

ISA
Model

RTL Model

Gate Level
Model

Prototype
Hardware

Software
Perf. Model
Arch Dep.
Proc Model

Source Code
HOL
Assembly

Load Module

[SASSP94]
Sanders Codesign Methodology

- Subsystems process
  - Processing requirements are modeled in an architecture-independent manner
  - Codesign not an issue
- Architecture process
  - HW/SW allocation analyzed via modeling of SW performance on candidate architectures
  - Hierarchical verification is performed using finer grain modeling (ISA and below)
- Detailed design
  - Downloadable executable application and test code is verified to maximum extent possible
- Library support
  - SW models validated on test data
  - HW models validated using existing SW models
  - HW & SW models jointed iterated throughout designs
Lockheed Martin ATL Codesign Methodology
**Major Codesign Research Efforts**

- **Chinook** - University of Washington - Chou, Ortega, Borriello
- **Cosmos** - Grenoble University - Ismail, Jerraya
- **Cosyma** - University of Braunschweig - Ernst, Henkel, Benner
- **Polis** - U. C. Berkeley - Chiodo, Giusto, Jurecska, Hsieh, Lavagno, Sangiovanni-Vincentelli
- **Ptolemy** - U. C. Berkeley - Kalavade, Lee
- **Siera** - U. C. Berkeley - Srivastava, Broderson
Chinook

- Unified representation: Event Graph (CDFG)
- Partitioning: constraint driven by scheduling requirements
- Scheduling: timing driven
- Modeling substrate: based on Verilog HDL
- Validation: simulation based (Verilog)
- Main emphasis on synthesis of hardware/software interfaces
Cosmos

- Unified representation: Initial description is done in SDL (specification description language) which is translated into SOLAR, an intermediate form that allows several description levels (CSPs, FSMs, etc.)
- Partitioning: user driven using a tool that allows processes to be grouped together or split into sub-processes
- Scheduling: based on the partitioning
- Modeling substrate: VHDL simulation after architecture mapping
- Validation: simulation based
- Main emphasis on synthesis of communications mechanisms between processes - reuse of existing communication models
Cosyma

• Unified representation: ES graph (CDFG)
• Partitioning: combined method based on course partitioning by user with cost guidance and finer scheduling done by simulated annealing
• Scheduling: no specific method
• Modeling substrate: based on C++
• Validation: simulation based (C++)
• Main emphasis on partitioning for hardware accelerators
Polis

- Unified representation: Codesign Finite State Machine (CFSM) based
- Partitioning: user driven with cost estimated provided by co-simulation
- Scheduling: classical real-time algorithms
- Modeling substrate: Ptolemy based (C++)
- Validation: co-simulation and formal FSM verification
- Main emphasis on verifiable specification not biased to either hardware or software implementation
Ptolemy

- Unified representation: Data Flow Graph
- Partitioning: greedy algorithm based on scheduling constraints
- Scheduling: linear based on sorting blocks by “criticality”
- Modeling substrate: heterogeneous modeling and simulation framework based on C++
- Validation: based on simulation
- Main emphasis on heterogeneous modeling framework (mixing different models of computation)
Siera

- Unified representation: static, hierarchical network of concurrent sequential processes communicating via message queues (similar to DFG)
- Partitioning: manual user driven
- Scheduling: static process to processor mapping, priority based preemptive schedulers available within real-time OS on processors
- Modeling substrate: based on VHDL - includes support for modeling continuous time systems such as sensors and actuators
- Validation: based on simulation
- Main emphasis on the design of embedded systems targeted towards a predefined architectural template
Chinook

- Hardware/Software Co-synthesis system developed at the University of Washington
- Targeted at real-time reactive embedded systems
- Control dominated designs constructed from off-the-shelf components
Chinook’s Principal Innovations

• Single Specification - one specification, with explicit timing/performance constraints is used for the system’s hardware and software
• One Simulation Environment - the high level specification, the final result, and any intermediate steps can be simulated to verify and debug the design
• Software Scheduling - the appropriate software architecture is synthesized to meet the timing requirements
• Interface Synthesis - the hardware and software necessary to interface between system components (glue logic and device drivers) is automatically synthesized
• Complete Information for Physical Prototyping - a complete netlist is generated for the hardware, and C source code is generated for the software
The Chinook System

Verilog Specification

Processor & Device Libraries

Behavioral Simulation

Mixed Simulation

Structural Simulation

Parser

Scheduler

Comm. Synthesizer

Code Generator

Driver Synthesizer

Interface Synthesizer

Netlist
System Specification in Chinook  
(Unified Representation)

• The system specification is written in a dialect of Verilog and includes the system’s behavior and the structure of the system architecture
• The behavior is specified as a set of tasks in a style similar to communicating finite state machines - control states of the system are organized as modes which are behavioral regimes similar to hierarchical states
• In a given mode, the system’s responses are defined by a set of handlers which are essentially event-triggered routines
• The designer must tag tasks or modules with the processor that is preferred for their implementation - untagged tasks are implemented in software
• The designer can specify response times and rate constraints for tasks in the input description
Scheduling in Chinook

- Chinook provides an automated scheduling algorithm
- Low-level I/O routines and high level routines grouped in modes are scheduled statically
- A static, nonpreemptive scheduling algorithm is used to meet min/max timing constraints on low-level operations
  - Determines serial ordering for operations
  - Inserts delays as necessary to meet minimum constraints
  - Includes heuristics in the scheduling algorithm to help exact algorithm generate valid solution to NP-hard scheduling problem
- A customized dynamic scheduler may be generated for the top-level modes
Interface Synthesis in Chinook

- Realization of communication between system components is an area of emphasis in the Chinook system
- Chinook synthesizes device drivers from timing diagrams
- Custom code for the processor being used is generated
  - For processors with I/O ports, an efficient heuristic is used to connect devices with minimal interface hardware
  - For processors w/o I/O ports, a memory mapped I/O interface is generated including allocating address spaces, and generating the required bus logic and instructions
- Portions of the interface that cannot be implemented in software are synthesized into external hardware
Communications Synthesis and System Simulation in Chinook

- Chinook provides methods for synthesizing communications systems between multiple processors if a multicomputer implementation is chosen
  - Bus-based, point-to-point, and hybrid communications schemes are supported
  - Communications library that includes FIFOs, arbiters, and interconnect templates is provided

- Simulation of the design at different levels of detail is supported
  - Verilog-XL Programming Language is used
  - Verilog PLI is used to interface to device models written in C
  - Each device supports the same API for simulation and synthesis - API calls can be used by the designer to animate the model interactively
  - RTL level models of the processors are used to simulate the final implementation of the system (software)
Cosynthesis of Embedded Applications (COSYMA)

- Developed at the Technical University of Braunschweig, Germany
- An experimental system for HW/SW codesign of small embedded real time systems
  - Implements as many operations as possible in software running on a processor core
  - Generates external hardware only when timing constraints are violated
- Target architecture:
  - Standard RISC processor core
  - Application-specific processor
- Communication between HW and SW through shared memory with a communicating sequential processes (CSP) type protocol
COSYMA (Cont.)

• Input description of system in C* is translated into an internal graph representation supporting
  – Partitioning
  – Generating hardware descriptions for parts moved to hardware

• Internal graph representation combines
  – Control and dataflow graph
  – Extended syntax (ES) graph
    • Syntax graph
    • Symbol table
    • Local data/control dependencies
Design Flow in a COSYMA System

- C* Mode
  - C* Compiler
    - ES Flowgraph
      - Partitioning
        - Cost Estimation
          - Run time Analysis
            - Simulator
              - ES to HW C
                - HW-C Model
                  - Olympus
                    - ES to C
                      - C Program
                        - C Compiler
                          - Object Code
COSYMA - Aims and Strategies

• Major aim is automating HW/SW partitioning process, for which very few tools currently exist
• COSYMA partitions at the basic block and function level (including hierarchical function calls)
  – Simulated annealing algorithm is used because of its flexibility in the cost function and the possibility to trade-off computation time vs result quality
  – Starts with an unfeasible all-software solution
COSYMA - Cost Function and Metrics

- The cost function is defined to force the annealing to reach a feasible solution before other optimization goals (e.g., area).
- The metrics used in cost computation are:
  - Expected hardware execution times
  - Software execution times
  - Communication
  - Hardware costs
- The cost function is updated in each step of the simulated annealing algorithm.
After partitioning, the parts selected to be realized in software are translated to a C program, thereby inserting code for communicating with the coprocessor.

The rest of the system is translated to the input description of the high-level synthesis system, and an application-specific coprocessor is synthesized.

Lastly, a fast-timing analysis of the whole HW/SW system is performed to test whether all constraints are satisfied.
Ptolemy

• A software environment for simulation and prototyping of heterogeneous systems

• Attributes
  – Facilitates mixed-mode system simulation, specification, and design
  – Supports generation of DSP assembly code from a block diagram description of algorithm
  – Uses object-oriented representations to model subsystems efficiently
  – Supports different design styles called domains
Codesign Methodology
Using Ptolemy

- Ptolemy supports a framework for hardware/software codesign, called the *Design Assistant*.

- The Design Assistant consists of two components:
  - Specific point tools for estimation, partitioning, synthesis, and simulation.
  - An underlying design methodology management infrastructure for design space exploration.
Codesign Methodology Using Ptolemy (Cont.)

- Design constraints
- Design specs.
- User inputs

Design Flow

Area/Time Estimation

HW/SW Partitioning

Hardware Synthesis

Interface Synthesis

Netlist Generation

VHDL/Synopsys

Software Synthesis

Simulation

System

Layout + Software

- CPLEX(ILP)
- GCLP
- Ptolemy
- Manual

© IEEE 1994
[Rozenblit94]
Ptolemy Heterogeneous Simulation Environment
Structural Components

• Data encapsulated in “particles”
• “Block” objects send and receive messages
• Particles travel to/from external world through “portholes”
POLIS

- Hardware/Software Codesign and synthesis system developed at the University of California, Berkeley
- Targeted towards small, scale, reactive, control dominated embedded systems
- Includes an “unbiased” mechanism for specifying the system’s function that allows for maximum flexibility in mapping to hardware or software and also allows for formal verification
System behavior is specified in a formal manner using Codesign Finite State Machines (CFSMs)
- CFSMs translate a set of inputs to a set of outputs with only a finite amount of internal state
- Unlike traditional FSMs, CFSMs do not all change state exactly at the same time (globally asynchronous)
CFSMs are designed to be unbiased towards hardware or software
Translators exist to convert other specification languages (e.g. ESTEREL) into CFSMs
CFSMs can be translated into traditional FSMs to allow formal verification
CFSMs can communicate with each other using events
- Events are unidirectional and happen in non-zero, unbounded time
- Events can be used to communicate across all domains (hardware or software)
- Events are unbuffered and can be overwritten - however, they can be used to implement fully interlocked handshaking
CFSMs are translated into behavioral FSMs for hardware synthesis and into S-graphs for software synthesis
Codesign Finite State Machines

- Specification: “Five seconds after the key is turned on, if the belt has not been fastened, an alarm will beep for ten seconds or until the key is turned off”
S-graph Software Specification
Partitioning and Scheduling in POLIS

• Partitioning based on mapping CFSMs to either hardware or software
• This mapping is left to the user - performance feedback is provided by simulation
• Interfaces between partitions are automatically generated
• Scheduling based on executing CFSMs
• Selection of scheduling algorithm left to user - built into RTOS
  – Round-robin cyclic executive
  – Off-line I/O rate-based cyclic executive
  – Static pre-emptive: rate monotonic scheduling
  – Dynamic pre-emptive: Earliest Deadline First
Interfaces Among Partitions

- Interfaces use strobe/data protocol (corresponding to the event/value primitive)

A
B
C

Sender

Receiver

Sender’s Domain

Channel’s Domain

Receiver’s Domain

Example HW to SW interface

HW

HW to SW

SW

X

y

ack

11 + 0- / 0
-1 / 0
-0 / 1
10 / 1

x ack / y

ack
The POLIS Co-design Environment
References


References (Cont.)


[IEEE] All referenced IEEE material is used with permission.


References (Cont.)

Additional Reading:


Final issues

• Come by my office hours (right after class)

• Any questions or concerns?