Codesign Framework

Parts of this lecture are borrowed from lectures of Johan Lilius of TUCS and ASV/LL of UC Berkeley.

-Rabi Mahapatra
Embedded Processor Types

• General Purpose
  - Expensive, requires more support logics, unpredictable

• Controllers
  - Control flow dominated, bit-level operation emphasized
  - Example: PXI EC from NI

• Field Programmable Gate Arrays
  - Configurable, replaces processors and drivers to itself

• DSP
  - Based upon multiply-add module to suit DSP algorithms
Domain specific processors

- Network processors
  - Array processors (16-256) to process internet packets (QuantumFlow of Cisco)
  - High-end routers: Ex: FP3 from Lucent

- Media processors
  - SIMD or vector processor (RISC, VLIW) like StarPro2700 (LSI), ARMADA1000 (Marvel)
  - Digital camera, DVD player etc.
System Design

What is System Design?

“Is the process of implementing a desired functionality using a set of components”
System Design contd.

Step 1

Design must begin with specifying the desired functionality
For precise specification, we need to
- think the system to be a collection of simpler subsystems
- a method (rules) to compose these pieces
Functions vs. Computation

• Functions specify only a relation between two sets of variables (input and output)
• Computation describe how the output variables can be derived from the value of input variable
Essential Issues

- Modeling
  - System description
  - System behavior
- Validation (verification)
- Performance estimation
- Partitioning and mapping
- Synthesis:
  - Software synthesis
  - Hardware synthesis
  - Interface synthesis
A Co-design framework

ASSUMPTION:
Homogenous model of computation
The method or rules to compose the pieces of subsystems to create a system functionality is usually called a Model.
Model should be…

- *formal*: no ambiguity
- *complete*: with sets of properties, performance indices and constraints
- *comprehensive* and easy to modify
- *natural* to understand
What is a model?

Model is a formal system consisting of objects and composition rules, and is used for describing a system’s characteristics.
Modeling
d
- Modeling digital systems is often complicated by the heterogeneity of its components.

- Distinguish between:
  - models of computation
  - hardware/software (specification) languages.

- A language may imply many models:
  - UML State Machines:
    - synchronous behavior within a state machine
    - asynchronous behavior between state machines
Model of Computation

• A MoC is a framework in which to express what sequence of actions must be taken to complete a computation
• An instance of a model of computation is a representation of a function under a particular interpretation of its constituents.
• Not necessarily a bijection (almost never!)
Models of Computation

• Often existing models belong to several categories
  – Finite state machines:
    • totally ordered discrete events
  – Petri nets:
    • partially ordered events
  – Synchronous data-flow:
    • multirate discrete time
    • partially-ordered events
why different Models?

- Various models have certain strong properties that might be useful for some applications
- Some problems might be un-decidable
Specification languages

- Finite-state based
  - Synchronous communication:
    - Statecharts, Esterel
  - Asynchronous communication:
    - SDL
- Partial orders of tasks:
  - VHDL at behavioral level
- Discrete time, cycle based:
  - VHDL at RTL level
- Synchronous data-flow: Silage, DFL, Lustre
Specification languages

• No perfect language exists!
  – Control (software): asynchronous (fsm)
  – DSP: data-flow
  – Hardware: synchronous

⇒ Force user to think in terms of:
  – one model of computation system: POLIS
  – many models of computation: PTOLEMY
A Co-design framework

ASSUMPTION:
Homogenous model of computation

System description
Compiler
System behavior
Partitionning and mapping
Partitioned specification

Evaluation
SW synthesis
Interface synthesis
Hw synthesis

Prototype
C-code
VHDL
VHDL

Validation
Performance estimation
Validation

• System-level validation (co-validation):
  – Methods for gaining reasonable certainty that the design is free from errors.

• Methods:
  – Verification
  – (Co-)Simulation
  – Emulation
Verification

• Specification verification
  • Is the specification consistent?
  • Does it have the required properties?

• Implementation verification
  • Have we implemented the specification?

• Checking of:
  • safety: nothing bad ever happens
  • liveness: something good eventually happens
Weakly heterogeneous systems

- One or more processors, some dedicated hardware, and several software layers.
- Desired features of simulator:
  - adequate timing accuracy
  - fast execution
  - visibility of internal registers for debugging purposes
- Problems:
  - One step in program is equivalent to many steps in hardware
    ⇒ long running times
    ⇒ Availability of hardware models with the required abstraction level.
Highly heterogeneous systems

• Specialized simulators:
  – PTOLEMY (U.C. Berkeley)
A Co-design framework

ASSUMPTION:
Homogenous model of computation

System description

Compiler

System behavior

Validation

Partitionning and mapping

Partitioned specification

Performance estimation

Evaluation

SW synthesis

Interface synthesis

Hw synthesis

C-code

VHDL

VHDL

Prototype

9/15/15
Partitioning

• Input: functional specification
• Output:
  – an architecture composed of
    • hardware black boxes,
    • software black boxes and
    • interconnection media and mechanisms
  – a **mapping function** that assigns functional units to architectural units
Partitioning

• Construction of mapping is an optimization problem:
  – mapping function optimizes
    • Cost, time, power, area, communication

• What is the architecture?
  – Automated synthesis of custom architectures difficult, common restrictions:
    • limited to a library of predefined choices
    • communication mechanisms are standardized
A Co-design framework

ASSUMPTION: Homogeneous model of computation

System description

Compiler

System behavior

Validation

Partitioning and mapping

Partitioned specification

Performance estimation

Evaluation

SW synthesis

Interface synthesis

Hw synthesis

C-code

VHDL

Prototype

9/15/15
Hardware synthesis

- Well established research field
- Several commercial tools exist
- Levels of abstraction:
  - Behavioral synthesis: algorithmic synthesis
  - Register-Transfer level synthesis: VHDL, Verilog
  - Logic level synthesis: netlist
- Research issue: reuse of hardware
Software synthesis

- Difficult problem for general purpose computing
- For embedded system much more constrained:
  - no swapping devices
  - no stacks
  - only polling and static variables
- Simple algorithms
  - Translating FSMs to programs especially simple
- Specification consists of concurrent tasks
  - Problem: How do we find a linear execution order that satisfies the timing constraints?
    - Use scheduling theory.
Interface synthesis

- Interface between processor and ASIC
  - synthesis of software
  - synthesis of "glue logic"
- Automatic generation of bus interfaces
  - PCI, VME
- Packet routing against wires
- Interfacing of sensors and actuators
Example Tools

- **Academic:**
  - POLIS: U.C. Berkeley
  - PTOLEMY: U.C. Berkeley
  - VULCAN: Stanford U. (Hardware C)
  - CHINOOK: U. of Washington (VHDL)
  - COSYMA: U. of Braunschweig (C*)
  - MEIJE: INRIA and others (Esterel, Lustre, Signal)

- **Commercial:**
  - Arexys: SDL, VHDL, C
  - CoWare: C/C++
  - LavalLogic: Java to Verilog
  - Cynlib: C++ to Verilog
  - Art, Algorithm to RT: C++ to RTL
  - SUPERLOG: System level description language
Available tools

• Synopsis Tool
• Carbon Design Studio
• PSOC
• Xilinx FPGA
• Xtensa from Tensilica (optional)
• VxWorks, Montavista RTOS (optional)
• Code Composer from TI (optional)
• Specification: FSM-based languages (Esterel,...)
• Internal Representation: CFSM network
• Validation:
  • high-level co-simulation
  • FSM-based formal verification
• Partitioning: by hand, based on co-simulation estimates
• Scheduling: classical real-time algorithms
• Synthesis:
  • S-graph-based code synthesis for software
  • logic synthesis for hardware
• Main emphasis on *unbiased verifiable specification*
- Specification: Data flow graph
- Internal representation: DFG
- Validation: multi-paradigm co-simulation (DF, discrete events, ...)
- Partitioning: greedy, based on scheduling
- Scheduling: linear, sorting blocks by ```criticality```
- Synthesis:
  - DSP code
  - custom DSP synthesis for hardware
- Main emphasis on heterogeneous computation models
Projects

- Power aware codesign – Low-Power Scheduling
- NoC, Communication synthesis in SoC
- IoT Framework and Security Design
- Data Analytic Accelerator
- Distributed Embedded systems
- Novel design methodology
- Your own project
Project (contd.)

- Pre-proposal
- Proposal
- Mid-term Report
- Final Report
Student Survey/Feedback

• Your strength (theory, hardware, software, analytical, based on past experiences and exposure.)

• Interests in ES area (topics, learning preferences, designs, case studies, methods, depth or breadth etc)

• Class activity preferences: (1 high, 4 low)
  – paper reviews,
  – hands on assignments,
  – projects,
  – presentations