Specifying and modeling languages

Introduction
Software oriented design representations
Hardware oriented design representations
Graph-based design representations
Resource descriptions

Available resources – System-level

General-purpose SW processors
- Digital signal processors (DSPs)
- Application-specific integrated circuits
  - Dynamically reconfigurable hardware
    - E.g., field-programmable gate arrays (FPGAs)
  - Busses
  - Wireless communication channels
  - Wires

Available resources – High-level

Simple arithmetic/logic units
- Multiplexers
- Registers
- Wires

Specification language requirements

Describe hardware (HW) and software (SW) requirements
Specify constraints on design
- Indicate system-level building blocks
  - To allow flexibility in synthesis, must be abstract
    - Differentiate HW from SW only when necessary
    - Concentrate on requirements, not implementation
    - Make few assumptions about platform

Software oriented design representations

ANSI-C
- Advantages
  - Huge code base
  - Many experienced programmers
  - Efficient means of SW implementation
  - Good compilers for many SW processors
- Disadvantages
  - Little implementation flexibility
    - Strongly SW oriented
    - Makes many assumptions about platform
  - Poor support for fine-scale HW synchronization

SystemC
- Advantages
  - Support from big players
    - Synopsys, Cadence, ARM, Red Hat, Ericsson, Fujitsu, Infineon Technologies AG, Sony Corp., STMicroelectronics, and Texas Instruments
- Disadvantages
  - Extension of SW language
  - Not designed for HW from the start
  - Compiler available for limited number of SW processors
  - New
Numerous competitors

Numerous languages
ANSI-C, C++, and Java are most popular starting points
In the end, few can survive
SystemC has broad support

VHDL

Advantages
Supports abstract data types
System-level modeling supported
Better support for test harness design
Disadvantages
Requires extensions to easily operate at the gate-level
Difficult to learn
Slow to code

Verilog

Advantages
Easy to learn
Easy for small designs
Disadvantages
Not designed to handle large designs
Not designed for system-level

Esterel

Easily allows synchronization among parallel tasks
Works above RTL
Doesn’t require explicit enumeration of all states and transitions
Recently extended for specifying datapaths and flexible clocking schemes
Amenable to theorem proving
Translation to RTL or C possible
Commercialized by Esterel Technologies

Graph based design representations

Dataflow graph (DFG)
Synchronous dataflow graph (SDFG)
Control flow graph (CFG)
Control dataflow graph (CDFG)
Finite state machine (FSM)
Petri net
Periodic vs. aperiodic
Real-time vs. best effort
Discrete vs. continuous timing
Example from research

• Nodes are tasks
• Edges are data dependencies
• Edges have communication quantities
• Used for digital signal processing (DSP)
• Often cyclic when real-time
• Can be cyclic when best-effort
### Synchronous dataflow graph (SDFG)

![Synchronous dataflow graph (SDFG)](image)

- **Nodes**: A, B, C
- **Arcs**: A→B, B→A, B→C, C→B
- **Weights**: 1, 2, 3, 4

### Control dataflow graph (CDFG)

![Control dataflow graph (CDFG)](image)

- **Nodes**: if, k = k + 1, j = j + 5
- **Arcs**: true, false
- **Weights**: 30 Kb, 387 Kb

### Finite state machine (FSM)

- **States**: 00, 01, 02, 03, 04, 05, 06, 07, 08, 09
- **Transitions**: 1, 0
- **Conditions**: if $i < 2$, if $k = 3$

### Petri net

![Petri net](image)

- **Places**: P1, P2, P3
- **Transitions**: T1, T2, T3
- **Tokens**: 0, 1

### Periodic graphs

- **Graphs**: M/D/3/2
- **Properties**: Markov arrival, deterministic service delay
- **Scheduling**: least common multiple of periods

---

**Extensions**

- **Supports conditionals, loops**
- **No communication quantities**
- **Used by high-level synthesis algorithms**
- **Extensions represent SW, e.g., CFSMs**

**Petri net**

- **Graph**: composed of places, transitions, and arcs
- **Tokens**: produced and consumed
- **Useful model**: for asynchronous and stochastic processes
- **Places**: can have priorities
- **Not well-suited**: for representing dataflow systems
- **Timing analysis**: quite difficult
- **Large flat graphs**: difficult to understand

**Periodic graphs**

- **Some system specifications**: contain periodic graphs
- **Can guarantee**: scheduling validity by scheduling to the least common multiple of periods
- **Aperiodic specifications**: can also meet, however, resources will sometimes be idle
Discrete vs. continuous timing

System-level: continuous
  Operations are not small integer multiples of the clock cycle
High-level: discrete
  Operations are small integer multiples of the clock cycle
Implications:
  System-level scheduling is more complicated...
  ...however, high-level also very difficult.

Real-time vs. best effort

Why make decisions about system implementation statically?
  Allows easy timing analysis, hard real-time guarantees
  If a system doesn’t have hard real-time deadlines, resources can be more efficiently used by making late, dynamic decisions
  Can combine real-time and best-effort portions within the same specification
    Reserve time slots
    Take advantage of slack when tasks complete sooner than their worst-case finish times

Processing resource description

Often table-based
  Price, area
  For each task
    Execution time
    Power consumption
    Preemption cost
    etc.
  etc.
  Similar characterization for communication resources
  Wise to use process-based
Communication resource description

Can use bus-bridge based models for distributed systems
Wireless models
etc.
However, in the future, it will become increasingly important to
draw SOC communication model on process parameters

Example from research

Data-flow graphs
Multirate
Hard real-time
Some aperiodic hard real-time tasks
May have best-effort tasks

System-level representations summary

No single representation has been decided upon
Software-based representations becoming more popular
System-level representations will become more important
This is still an active area of research

Notes on clustering and partitioning

Interdependence with architecture
Heterogeneity’s impact on partitioning
Applications to grid computing
Dynamic partitioning

Interesting future direction

Open problem

Can specification be so simple for some embedded application
domains that application experts who are not computer engineers
easily do it?
What HCI, compiler, and synthesis support is required?

Clustering and system-level pipelining references

Steven Edwards, Luciano Lavagno, Edward A. Lee, and Alberto
Sangiovanni-Vincentelli. Design of embedded systems: Formal
models, validation, and synthesis. Proc. IEEE, (3):366–390,
March 1997
Recent technical report from embedded systems languages expert
Wayne Wolf. Embedded computing systems and
hardware/software co-design. In Wai-Kai Chen, editor, The VLSI
Handbook. CRC Press, 2006
Assignment: Write a short paragraph describing the most important
points in both of these articles.

Reading assignment

Engineering, Princeton University, July 2002: Chapter 4
Definitions and introduction to stochastic optimization

Decide topics and groups for project

Determine counts for each, add additional topics

- Models and languages
- Formal methods for designing reliable embedded systems
- Reliability optimization
- Heterogeneous multiprocessor synthesis
- Real-time systems
- Scheduling
Decide topics and groups for project

- Compilation techniques for embedded systems
- Embedded operating systems
- Low-power and power-aware design
- Low-power and power-aware design (continued)
- Novel fabrication techniques for compact and low-power embedded systems
- Emerging applications: sensor networks
- Hardware and software data compression for use in embedded systems

Next class

Overview of optimization techniques
Example solution to heterogeneous MPSoC synthesis problem
Survey of formal methods for designing reliable embedded systems