SPECIFICATION AND DESIGN OF EMBEDDED SYSTEMS

by

Daniel D. Gajski
Frank Vahid
Sanjiv Narayan
Jie Gong

University of California at Irvine
Department of Computer Science
Irvine, CA 92715-3425
Design representations

- **Behavioral**
  Represents functionality but not implementation

- **Structural**
  Represents connectivity but not dimensionality

- **Physical**
  Represents dimensionality but not functionality
# Levels of abstraction

<table>
<thead>
<tr>
<th>Levels</th>
<th>Behavioral forms</th>
<th>Structural components</th>
<th>Physical objects</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transistor</td>
<td>Differential eq., current–voltage diagrams</td>
<td>Transistors, resistors, capacitors</td>
<td>Analog and digital cells</td>
</tr>
<tr>
<td>Gate</td>
<td>Boolean equations, finite–state machines</td>
<td>Gates, flip–flops</td>
<td>Modules, units</td>
</tr>
<tr>
<td>Register</td>
<td>Algorithms, flowcharts, instruction sets,</td>
<td>Adders, comparators, registers, counters, register files, queues</td>
<td>Microchips, ASICs</td>
</tr>
<tr>
<td></td>
<td>generalized FSM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Processor</td>
<td>Executable spec., programs</td>
<td>Processors, controllers, memories, ASICs</td>
<td>PCBs, MCMs</td>
</tr>
</tbody>
</table>
Design methodologies

- Capture-and-simulate
  Schematic capture
  Simulation

- Describe-and-synthesize
  Hardware description language
  Behavioral synthesis
  Logic synthesis

- Specify-explore-refine
  Executable specification
  Software and hardware partitioning
  Estimation and exploration
  Specification refinement
Motivation

Executable specification

if \(x = 0\) then
\[\begin{align*}
y &= a \times b / 2
\end{align*}\]

System implementation

Processor

Memory

Video accelerator

ASIC

I/O

Models
Languages

Partitioning
Estimation
Refinement

Software compilation
Behavioral synthesis
Logic synthesis

Physical design
Test generation
Manufacturing
Outline

- Introduction
- Design models and architectures
- System-design languages
- An example
- Translation
- Partitioning
- Estimation
- Refinement
- Methodology and environments
Models and architectures

Models are conceptual views of the system’s functionality
Architectures are abstract views of the system’s implementation
Models and architectures

- Model: a set of functional objects and rules for composing these objects
- Architecture: a set of implementation components and their connections
Models of an elevator controller

"If the elevator is stationary and the floor requested is equal to the current floor, then the elevator remains idle.
If the elevator is stationary and the floor requested is less than the current floor, then lower the elevator to the requested floor.
If the elevator is stationary and the floor requested is greater than the current floor, then raise the elevator to the requested floor."

(a) English description

(b) Algorithmic model

loop
  if (req_floor = curr_floor) then
    direction := idle;
  elsif (req_floor < curr_floor) then
    direction := down;
  elsif (req_floor > curr_floor) then
    direction := up;
  end if;
end loop;

(c) State-machine model
Architectures for implementing the elevator controller

(a) Register level

(b) System level
Models

- **State-oriented models**
  - Finite-state machine (FSM), Petri net, Hierarchical concurrent FSM

- **Activity-oriented models**
  - Data flowgraph, Flowchart

- **Structure-oriented models**
  - Block diagram, RT netlist, Gate netlist

- **Data-oriented models**
  - Entity-relationship diagram, Jackson’s diagram

- **Heterogeneous models**
  - Control/data flowgraph, Structure chart, Programming language paradigm, Object-oriented paradigm, Program-state machine, Queueing model
State oriented: Finite-state machine (Mealy model)

- States: $S = \{s_1, s_2, s_3\}$
- Inputs: $I = \{r_1, r_2, r_3\}$
- Outputs: $O = \{d_2, d_1, n, u_1, u_2\}$

Transition function $f: S \times I \rightarrow S$
Output function $h: S \times I \rightarrow O$
State oriented: Finite-state machine (Moore model)
State oriented: Finite-state machine with datapath

\[ S_1 \]

(start)

\[ \text{if } (\text{curr\_floor} \neq \text{req\_floor}) \text{ then } \text{output} := \text{req\_floor} - \text{curr\_floor}; \text{curr\_floor} := \text{req\_floor} \]

\[ \text{if } (\text{curr\_floor} = \text{req\_floor}) \text{ then } \text{output} := 0 \]
Finite-state machines

- **Merits:**
  - represent system’s temporal behavior explicitly
  - suitable for control-dominated system

- **Demerits:**
  - lack of hierarchy and concurrency resulting in
  - state or arc explosion when representing complex systems
State oriented: Petri nets

Net = (P, T, I, O, u)

P = \{p1, p2, p3, p4, p5\}
T = \{t1, t2, t3, t4\}

I(t1) = \{p1\}
I(t2) = \{p2, p3, p5\}
I(t3) = \{p3\}
I(t4) = \{p4\}

O(t1) = \{p5\}
O(t2) = \{p3, p5\}
O(t3) = \{p4\}
O(t4) = \{p2, p3\}

u: u(p1) = 1  
u(p2) = 1  
u(p3) = 2  
u(p4) = 0  
u(p5) = 1
Petri nets

(a) Sequence

(b) Branch

(c) Synchronization

(d) Resource contention

(e) Concurrency
Petri nets

- **Merits:**
  good at modeling and analyzing concurrent systems

- **Demerits:**
  ‘flat’ model that is incomprehensible when system complexity increases
State oriented: Hierarchical concurrent FSM
Hierarchical concurrent FSMs

- **Merits:**
  - support both hierarchy and concurrency
  - good for representing complex systems

- **Demerits:**
  - concentrate only on modeling control aspects
  - and not data and activities
Activity oriented: Data flowgraphs (DFG)

(a) Activity level

(b) Operation level
Data flowgraphs

- **Merits:**
  - support hierarchy
  - suitable for specifying complex transformational systems
  - represent problem-inherent data dependencies

- **Demerits:**
  - do not express temporal behaviors or control sequencing
  - weak for modeling embedded systems
Activity oriented: Flowchart (CFG)

start

\[ J = 1 \]
\[ MAX = 0 \]

\[ J = J + 1 \]

\[ J > N \]

MEM(J) > MAX

MAX = MEM(J)

end
Flowcharts

- **Merits:**
  - useful to represent tasks governed by control flow
  - can impose an order to supersede natural data dependencies

- **Characteristics:**
  - used only when the system’s computation is well known
Structure oriented: Component-connectivity diagrams

(a) Block diagram

Processor

Program memory

I/O coprocessor

Data memory

Application specific hardware

System bus

(b) RT netlist

Register file

LIR

RIR

ALU

Left bus

Right bus

(c) Gate netlist

A

B
Component-connectivity diagrams

- Merits:
  good at representing system’s structure

- Characteristics:
  often used in the later phases of design process
Data oriented: Entity-relationship diagram

- Availability
  - Supplier
  - P.O. instance
  - Product
- Customer
- Request
- Order
Entity-relationship diagrams

- **Merits:**
  provide a good view of the data in the system, also suitable for expressing complex relations among various kinds of data

- **Demerits:**
  do not describe any functional or temporal behavior of the system.
Data oriented: Jackson’s diagram

```
Drawing
  AND
  Color   Shape   Users *
    OR
    Circle   Rectangle
     AND
     Radius   Width   Height
```

Users
Jackson’s diagrams

- **Merits:**
  suitable for representing data having a complex composite structure.

- **Demerits:**
  do not describe any functional or temporal behavior of the system.
Heterogeneous: Control/data flowgraph

(a) Activity level

(b) Operation level
Control/data flowgraphs

- Merits:
  correct the inability of DFG in representing the control of a system
  correct the inability of CFG to represent data dependencies
Heterogeneous: Structure chart

- Control Data
- Get A, B
- Transform A, B
- Change_A A', B'
- Change_B A, B
- Compute A', B', C, D
- Out_C
- Do_Loop1
- Do_Loop2

Branch

Iteration
Structure charts

- **Merits:**
  - represent both data and control

- **Characteristics:**
  - used in the preliminary stages of program design
Heterogeneous: Programming languages

- Imperative vs declarative programming languages:
  C, Pascal, Ada, C++, etc.
  LISP, PROLOG, etc.

- Sequential vs concurrent programming languages:
  Pascal, C, etc.
  CSP, ADA, VHDL, etc.
Programming languages

- Merits:
  model data, activity, and control

- Demerits:
  do not explicitly model the system’s states
Heterogeneous: Object-oriented paradigm

[Diagram showing object-oriented paradigm with data and operations connected through transformation function]
Object-oriented paradigms

- **Merits:**
  support information hiding, inheritance, natural concurrency

- **Demerits:**
  not suitable for systems with complicated transformation functions
variable A: array[1..20] of integer

variable i, max: integer;

max = 0;
for i = 1 to 20 do
    if (A[i] > max) then
        max = A[i];
    end if;
end for
Program-state machines

- Merits:
  - represent system’s states, data, control and activities in a single model
  - overcome the limitations of programming languages and HCFSM models
Heterogeneous: Queueing model

(a) One server

(b) Multiple servers
Queueing model

- Characteristics:
  used for analyzing system’s performance, and can find utilization, queueing length, throughput
Applications

- Application-specific architectures
  Controller architecture,
  Datapath architecture,
  Finite-state machine with datapath (FSMD).

- General-purpose processors
  Complex instruction set computer (CISC)
  Reduced instruction set computer (RISC)
  Vector machine
  Very long instruction word computer (VLIW)

- Parallel processors
Controller architecture

- State register
- Next-state function
- Output function
- Inputs
- Outputs
Datapath architecture

(a) Three stage pipeline

(b) Four stage pipeline
FSMD

Datapath inputs

Datapath outputs

Control unit

State register

Next-state function

Output function

Datapath

Datapath inputs

Control

Status

Datapath outputs
CISC architecture

Control unit

Microprogram memory

MicroPC

Address selection logic

Instruction reg.

Datapath

Memory

PC

Control

Status
RISC architecture

- Control unit
  - Instruction reg.
  - Hardwired output and next-state logic
- Memory
  - Datapath
    - Register file
    - ALU
    - Data cache
  - Datapath
    - State register
    - Instr. cache
    - Control
    - Status
- Control unit
  - Memory
Vector machines

- Interleaved memory
  - Memory pipes
  - Memory pipes
  - Vector registers
  - Scalar registers
  - Vector functional unit
  - Scalar functional unit
Parallel processors: SIMD/MIMD

(a) Message passing

(b) Shared memory
Conclusion

- Different models focus on different aspects
- Proper model needs to represent system’s features
- Models are implemented in architectures
- Smooth transformation of models to architectures increases productivity