Modeling Embedded Systems

Tajana Simunic Rosing
Department of Computer Science and Engineering
University of California, San Diego.
ES Design

Verification and Validation

Hardware components

Hardware

Software components
Models, Languages and Tools

- Models: $M_1$, $M_2$
- Languages: $L_1$, $L_2$
- Tools
- Behavior

- State machine
- Sequent. program
- Data-flow
- Concurrent processes

- Verilog
- C/C++
- Java
- VHDL

- Implementation A
- Implementation B
- Implementation C
Elements of model of computation

- **Language:**
  - Synchronous vs. asynchronous languages

- **State**

- **Decidability**
  - Can a property be determined in a finite amount of time?
## Modes of Communication

<table>
<thead>
<tr>
<th></th>
<th>Transmitters</th>
<th>Receivers</th>
<th>Buffer Size</th>
<th>Blocking Reads</th>
<th>Blocking Writes</th>
<th>Single Reads</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unsynchronized</td>
<td>many</td>
<td>many</td>
<td>one</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>Read-Modify-write</td>
<td>many</td>
<td>many</td>
<td>one</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>Unbounded FIFO</td>
<td>one</td>
<td>one</td>
<td>unbounded</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>Bounded FIFO</td>
<td>one</td>
<td>one</td>
<td>bounded</td>
<td>no</td>
<td>maybe</td>
<td>yes</td>
</tr>
<tr>
<td>Single Rendezvous</td>
<td>one</td>
<td>one</td>
<td>one</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Multiple Rendezvous</td>
<td>many</td>
<td>many</td>
<td>one</td>
<td>no</td>
<td>no</td>
<td>yes</td>
</tr>
</tbody>
</table>
Communication

- **Message Passing**
  - Non-blocking
  - Blocking
  - Extended rendezvous

- **Shared memory**

```plaintext
process a {
  ..
  P(S) //obtain lock
  .. // critical section
  V(S) //release lock
}

process b {
  ..
  P(S) //obtain lock
  .. // critical section
  V(S) //release lock
}
```
Implantable Cardioverter Defibrillator

- **Function:**
  - helps stop dangerously fast heart rhythms in the heart's lower chambers
  - treats sudden cardiac arrest and restores a normal heartbeat.

- **Guarantee device works 100% of the time for at least 7-10yrs**
Models of Computation

- State Machine Models
  - FSM, CFSM, StateCharts, SDL
- Petri nets
- Communicating Processes
  - Kahn processes, Communicating Sequential Processes
- Ada
- Dataflow models
  - DFG, SDFG
- Discrete Event Systems
  - VHDL, Verilog, SystemC, SpecC
- Synchronous languages
  - Cycle based models, Esterel, Lustre
Classical automata

• Moore-automata:
  \( O = H(S); \quad S^+ = f(I, S) \)

• Mealy-automata
  \( O = H(I, S); \quad S^+ = f(I, S) \)
Co-design Finite State Machines (CFSMs)

- CFSm is FSM extended with:
  - Data handling
  - Asynchronous communication
- CFSM has
  - FSM part
  - Data computation part
- Interacting CFSMs communicate through broadcast events
Example CFSM Specification

- State of a CFSM includes those events that are at the same time input and output for it.
Network of CFSMs

- GALS model, no hierarchy
“Move the elevator either up or down to reach the requested floor. Once at the requested floor, open the door for at least 10 seconds, and keep it open until the requested floor changes. Ensure the door is never open while moving. Don’t change directions unless there are no higher requests when moving up or no lower requests when moving down…”
Elevator controller using a sequential program model

**Sequential program model**

Inputs: int floor; bit b1..bN; up1..upN-1; dn2..dnN;
Outputs: bit up, down, open;
Global variables: int req;

void UnitControl()
{
    up = down = 0; open = 1;
    while (1) {
        while (req == floor);
        open = 0;
        if (req > floor) { up = 1; }
        else { down = 1; }
        while (req != floor);
        up = down = 0;
        open = 1;
        delay(10);
    }
}

void RequestResolver()
{
    while (1)
        req = ...
}

void main()
{
    Call concurrently:
    UnitControl() and RequestResolver()
}

You might have come up with something having even more if statements.
Finite-state machine (FSM) model

*UnitControl process using a state machine*

![Finite-state machine diagram]

- **GoingUp** (req > floor)
  - u, d, o, t = 1, 0, 0, 0
  - !(req > floor)

- **Idle** (req == floor)
  - u, d, o, t = 0, 0, 1, 0
  - !(timer < 10)

- **GoingDn** (req < floor)
  - u, d, o, t = 0, 1, 0, 0

- **DoorOpen** (timer < 10)
  - u, d, o, t = 0, 0, 1, 1

u is up, d is down, o is open

- t is timer_start
**UnitControl with FireMode**

- **FireMode**
  - When *fire* is true, move elevator to 1st floor and open door
StateCharts: Hierarchy
Back to Elevator Example

UnitControl

NormalMode

FireMode

ElevatorController

RequestResolver

...
StateCharts: Default state
StateCharts: History
History & default state

same meaning
StateCharts: Concurrency

![Statechart Diagram]

- **answerings-machine**
  - on
  - line-monitoring
    - ring
    - hangup (caller)
  - Lwait
  - Lproc
  - key-monitoring (excl. on/off)
    - key pressed
    - done
  - Kwait
  - Kproc
  - key-on
  - key-off
  - off
Conditional Transitions
StateCharts: Timers
Example: Answering machine

- **Lproc**
- **4 s**
- **timeout**
- **play text**
- **beep**
- **talk**
- **return (callee)**
- **dead**
- **timeout**
- **silent**
- **beep**

- **8 s record**
Propagations and Broadcasts
StateCharts: Simulation

Status = values of all variables + set of events + current time
Step = execution of the three phases
StateCharts: edge labels

event [condition] / reaction

swap

\[ e/a := b \]

\[ e/b := a \]

\[ a := 1; b := 0 \]
StateCharts: Application Examples

- Power converter system for trams, metros & trains
  - System-level modeling and automated code generation
  - Guarantee latencies less than 10 microseconds
  - Cut development time by 50%
  - Time from design completion to first prototype down to 1hr from 3 months
  - Defect free code automatically generated for a number of RTOS implementations
StateCharts: Application Examples

- Development of defibrillator and pacemaker technology
  - Model system behavior before requirements are finalized
  - Check that SW provides mathematically consistent representation of the product’s system – correct and unambiguous representation of model behavior
  - More accurate and extensive verification – guarantee device works 100% of the time for at least 7-10yrs
  - Cut product verification costs by 20%
  - 15-20% overall cost reduction per projects on future development
StateCharts: Application Examples

- Jet engine electronic controller design
  - System level specification and consistency check
  - Construction of on-screen simulation of the cockpit display
  - Evaluate correctness in normal and faulty operation modes
- Project so successful that now StateCharts are used for:
  - Independent overspeed protection system
  - Thrust reverse control system
  - Engine fuel control system

By BMW
StateCharts: Summary

- Hierarchy
  - AND- and OR-super states
  - Default state
  - History
- Timing behavior
- State oriented behavior
- Edge labels
- Concurrency
- Synchronization & communication
  - Broadcast, shared memory
- Simulation
- Cross compiling
Models of Computation

- State Machine Models
  - FSM, CFSM, StateCharts, SDL
- Petri nets
- Communicating Processes
  - Kahn processes, Communicating Sequential Processes
- Ada
- Dataflow models
  - DFG, SDFG
- Discrete Event Systems
  - VHDL, Verilog, SystemC, SpecC
- Synchronous languages
  - Cycle based models, Esterel, Lustre
Data modem design

- What are the design goals?
- What should the design process be like?
SDL - Specification and Description Language
representation of FSMs/processes

http://www.sdl-forum.org/sdl88tutorial/
SDL - Operations on data

DCL
Counter Integer;
Date String;

Counter := Counter + 3;

Counter

(1:10) (11:30) ELSE
SDL- Communication among FSMs

Message passing via FIFO
SDL - Process interaction diagrams

BLOCK B1

Signal A.B;

process P1

Sw2 [A]

[A,B]

Sw1

process P2
SDL - Designation of recipients

Counter
TO OFFSPRING

Counter
Via Sw1

process P1

Sw2 [A]

[A,B] Sw1

process P2
SDL - Hierarchy
SDL - Timers

Process S

A -> g -> w -> B
B -> h -> x -> E
C -> i -> y -> D
D -> j -> set(now+p,T)

Timer T:
E -> f -> T
T -> v -> E
E -> v -> RESET(T)
E -> v -> A

RESET(T)
RESET(T)
SDL Application: Description of network protocols

System

Processor A  Router  Processor B  Processor C

C1  C2  C3

Block Processor A

layer-n

......

layer-1

Block Router

layer-n

layer-2

layer-1

Block Processor B

layer-n

......

layer-1

Block Processor C

layer-n

......

layer-1
SDL: Application Example

- ADSL design
  - Ideal language for telecom design; communication between components and their different states of operation can be easily modeled with SDL
  - Object orientation and automatic code generation significantly simplified system design verification
  - Early testing done on SDL – significant savings in the cost of expensive test equipment
SDL Summary

- SDL
  - Representation of processes
  - Communication & block diagrams
    - Unbounded FIFO
  - Timers and other language elements
  - Great for network protocols
Modeling Embedded Systems

Tajana Simunic Rosing
Department of Computer Science and Engineering
University of California, San Diego.
Models of Computation

- State Machine Models
  - FSM, StateCharts, SDL, CFSM
- Petri nets
- Communicating Processes
  - Kahn processes, Communicating Sequential Processes
- Ada
- Dataflow models
  - DFG, SDFG
- Discrete Event Systems
  - VHDL, Verilog, SystemC, SpecC
- Synchronous languages
  - Cycle based models, Esterel, Lustre
Petri net definitions

A Petri net is a 5-tuple, $PN = (P, T, F, W, M_0)$ where:

- $P = \{p_1, p_2, \ldots, p_m\}$ is a finite set of places,
- $T = \{t_1, t_2, \ldots, t_n\}$ is a finite set of transitions,
- $F \subseteq (P \times T) \cup (T \times P)$ is a set of arcs (flow relation),
- $W: F \rightarrow \{1, 2, 3, \ldots\}$ is a weight function,
- $M_0: P \rightarrow \{0, 1, 2, 3, \ldots\}$ is the initial marking,
- $P \cap T = \emptyset$ and $P \cup T \neq \emptyset$.

A Petri net structure $N = (P, T, F, W)$ without any specific initial marking is denoted by $N$.

A Petri net with the given initial marking is denoted by $(N, M_0)$.

---

### Input Places | Transition | Output Places
--- | --- | ---
Preconditions | Event | Postconditions
Input data | Computation step | Output data
Input signals | Signal processor | Output signals
Resources needed | Task or job | Resources released
Conditions | Clause in logic | Conclusion(s)
Buffers | Processor | Buffers

---

Example

- $H_2O$
Petri net – Train tracks model

- Train wanting to go right
- Train going to the right
- Track available
- Train going to the left
Concurrency, Causality, Choice
Concurrency, Causality, Choice
Concurrency, Causality, Choice
Concurrency, Causality, Choice
Concurrency, Causality, Choice
Petri nets: confusion 😊

- Concurrency & conflict lead to confusion….
  - Symmetric
  - Asymmetric

Source: Murata’89
Conflict for resource „track“

train wanting to go right

train going to the right

track available

train going to the left
Communication Protocol
Communication Protocol
Communication Protocol
Communication Protocol
Communication Protocol

![Diagram showing a communication protocol between P1 and P2 with states for sending and receiving messages and acknowledgments.]
Communication Protocol
Producer-Consumer Problem
Producer-Consumer Problem
Producer-Consumer Problem
Producer-Consumer Problem
Producer-Consumer Problem
Producer-Consumer Problem
Producer-Consumer Problem
Producer-Consumer Problem

![Diagram of Producer-Consumer Problem]
Producer-Consumer Problem
Producer-Consumer Problem
Producer-Consumer Problem
Producer-Consumer Problem
Producer-Consumer Problem
Producer-Consumer Problem
Producer-Consumer with Priority

- Modeling synchronization control when sharing resources
- Multiprocessor CPUs, distributed systems
A Petri net is a 5-tuple, $PN = (P, T, F, W, M_0)$ where:

- $P = \{p_1, p_2, \cdots, p_m\}$ is a finite set of places,
- $T = \{t_1, t_2, \cdots, t_n\}$ is a finite set of transitions,
- $F \subseteq (P \times T) \cup (T \times P)$ is a set of arcs (flow relation),
- $W: F \rightarrow \{1, 2, 3, \cdots\}$ is a weight function,
- $M_0: P \rightarrow \{0, 1, 2, 3, \cdots\}$ is the initial marking,
- $P \cap T = \emptyset$ and $P \cup T \neq \emptyset$.

A Petri net structure $N = (P, T, F, W)$ without any specific initial marking is denoted by $N$.

A Petri net with the given initial marking is denoted by $(N, M_0)$.

---

<table>
<thead>
<tr>
<th>Input Places</th>
<th>Transition</th>
<th>Output Places</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preconditions</td>
<td>Event</td>
<td>Postconditions</td>
</tr>
<tr>
<td>Input data</td>
<td>Computation step</td>
<td>Output data</td>
</tr>
<tr>
<td>Input signals</td>
<td>Signal processor</td>
<td>Output signals</td>
</tr>
<tr>
<td>Resources needed</td>
<td>Task or job</td>
<td>Resources released</td>
</tr>
<tr>
<td>Conditions</td>
<td>Clause in logic</td>
<td>Conclusion(s)</td>
</tr>
<tr>
<td>Buffers</td>
<td>Processor</td>
<td>Buffers</td>
</tr>
</tbody>
</table>

---

Example

Source: Murata’89
Petri Net Properties

- **Behavioral**
  - **Reachability**
    - Marking M reachable from marking Mo
  - **k - Boundedness**
    - Number of tokens in each place does not exceed finite number k
    - Safe if 1-bounded
  - **Liveness**
    - Can fire any transition of the net – related to absence of deadlocks

- **Structural**
  - **Controlability**
    - Any marking can be reached from any other marking
  - **Structural boundedness**
  - **Conservativeness** – weighted sum of tokens constant
PN Properties - Reachability

\[ M_0 = (1,0,1,0) \]
\[ M = (1,1,0,0) \]

\[ M_0 = (1,0,1,0) \]
\[ t_3 \]
\[ M_1 = (1,0,0,1) \]
\[ t_2 \]
\[ M = (1,1,0,0) \]
PN Properties - Liveness
PN Properties - Liveness

Not live
PN Properties - Liveness

Deadlock-free
PN Properties - Liveness

Deadlock-free
PN Properties - Boundedness
PN Properties - Boundedness
PN Properties - Boundedness
PN Properties - Boundedness
PN Properties - Boundedness
PN Properties - Conservation
PN Properties - Conservation
PN Properties - Conservation
Petri Nets - Analysis

- Structural
  - Incidence matrix
  - T- and S- Invariants
- State Space Analysis techniques
  - Coverability Tree
  - Reachability Graph
PN Properties – Analysis

Incident Matrix

State Equations
Petri Nets – Coverability Tree
Example

- Assume
  - $e=6$
  - $Mo = [0 \ 12]$
- Can we reach $M = [6 \ 0]$ from $Mo$?
- Can it be statically scheduled?
- What size buffers are needed at $P1$ & $P2$?
Petri nets: Applications

- Model, simulate and analyze networking protocols (e.g. TCP, Ethernet, etc)
- Model, simulate and analyze complex network elements (e.g. router, switch, optical mux); check their logical behavior
- Design and analyze network performance and AoS with logical models of traffic generators, protocols and network elements
- Study network behavior characteristics (e.g. throughput, blocking probability etc)
Petri nets in practice

Example apps:
- TCP performance
- Security system design and automated code generation
- MAC design
- ...

Tajana Simunic Rosing
Petri Nets - Summary

- PN Graph
  - places (buffers), transitions (action), tokens (data)
- Firing rule
  - Transition enabled if enough tokens in a place
- Properties
  - Structural (consistency, structural boundedness)
  - Behavioral (reachability, boundedness, etc.)
- Analysis techniques
- Applications
  - Modeling of resources, mutual exclusion, synchronization
Models of Computation

- State Machine Models
  - FSM, StateCharts, SDL, CFSM
- Petri nets
- Communicating Processes
  - Communicating Sequential Processes, Ada, Kahn processes
- Dataflow models
  - DFG, SDFG
- Discrete Event Systems
  - VHDL, Verilog, SystemC, SpecC
- Synchronous languages
  - Cycle based models, Esterel, Lustre
- HW Models
- Unified Modeling Language (UML)
Process

- **A sequential program**
  - Executes concurrently with other processes

- **Basic operations on processes**
  - Create & terminate, suspend & resume, join

- **Process communication**
  - Shared memory (and mutexes)
  - Message passing
  - Rendezvous
**Concurrent processes & implementation**

**Heartbeat Monitoring System**

- **Task 1:**
  - Read pulse
  - If pulse < Lo then
  - Activate Siren
  - If pulse > Hi then
  - Activate Siren
  - Sleep 1 second
  - Repeat

- **Task 2:**
  - If B1/B2 pressed then
  - Lo = Lo +/- 1
  - If B3/B4 pressed then
  - Hi = Hi +/- 1
  - Sleep 500 ms
  - Repeat

**Set-top Box**

- **Task 1:**
  - Read Signal
  - Separate Audio/Video
  - Send Audio to Task 2
  - Send Video to Task 3
  - Repeat

- **Task 2:**
  - Wait on Task 1
  - Decode/output Audio
  - Repeat

- **Task 3:**
  - Wait on Task 1
  - Decode/output Video
  - Repeat

**Communication Bus**

- **Processor A**
- **Processor B**
- **Processor C**
- **Processor D**

**General Purpose Processor**

(a) Processor A, Processor B, Processor C, Processor D

(b) General Purpose Processor

(c) General Purpose Processor
ADA-rendez-vous

task screen_output is
  entry call_ch(val:character; x, y: integer);
  entry call_int(z, x, y: integer);
end screen_out;
task body screen_output is
...
select
  accept call_ch ... do ..
end call_ch;
or
  accept call_int ... do ..
end call_int;
end select;

Sending a message:
begin
  screen_out.call_ch('Z',10,20);
exception
  when tasking_error =>
    (exception handling)
end;

Sending a message:
begin
  screen_out.call_ch('Z',10,20);
exception
  when tasking_error =>
    (exception handling)
end;
Task graphs

Nodes are a „program“ described in some programming language
Task graphs - Timing

Arrival time $[0,7]$  deadline $[1,8]$  deadline $[3,10]$

$T_1$  $T_2$  $T_3$
Task graphs - I/O
Task graphs - Shared resources

T1 \rightarrow T2 \rightarrow T3 \rightarrow T4 \rightarrow T5
Task graphs - Periodic schedules

... \( J_{n-1} \rightarrow J_n \rightarrow J_{n+1} \)

.. infinite task graphs
Task graphs - Hierarchy
Models of Computation

- State Machine Models
  - FSM, StateCharts, SDL, CFSM
- Petri nets
- Communicating Processes
  - Communicating Sequential Processes, Ada
- Dataflow models
  - Kahn processes
  - DFG, SDFG
- Discrete Event Systems
  - VHDL, Verilog, SystemC, SpecC
- Synchronous languages
  - Cycle based models, Esterel, Lustre
- Petri nets
- HW Models
- Unified Modeling Language (UML)
Kahn process network

KPN - executable task graphs
Communication is via infinitely large FIFOs
Nonblocking write, blocking read => DETERMINATE

Important properties of KPN:
Continuous
Output signals can be gradually produced; never have to consume all input to produce some output

Monotonic
Output depends on input but doesn’t change previous output

Continuous monotonic processes => ITERATIVE
Petri net model of a Kahn process

- KPNs are deterministic:
  - Output determined by
    - Process, network, initial tokens
Dataflow Process Networks

- **Dataflow:**
  - Maps input tokens to output tokens
  - Outputs function of current inputs
    - No need to keep state on suspend/resume
  - Scheduling for resource sharing

\[ Z = (A + B) \ast (C - D) \]

Nodes with arithmetic transformations:

Nodes with more complex transformations:
Dataflow process examples

- HW resource scheduling
  - Constraints: 2 mult, 2 ALU
  - List scheduler used
  - Additional constraints
    - Performance
    - Power
    - Area etc.

![Diagram showing dataflow process examples with nodal operators and data points.](image)
Synchronous dataflow
SDF notation

- Nodes have rates of data production or consumption.
- Edges have delays.
  - Delays do not change rates, only the amount of data stored in the system at startup.
What Latency & Sample Period can a SDFG achieve?

- $T_L = \max(I-O \text{ path}, D-O \text{ path})$
- $T_S = \max(D-D \text{ path}, I-D \text{ path})$
SDFG can be Transformed to Affect $T_L$ & $T_S$
SDF examples
SDF Scheduling

By building a set of “flow and conservation” equations

$3a - 2b = 0$
$4b - 3d = 0$
$b - 3c = 0$
$2c - a = 0$
$d - 2a = 0$

Solution: $a = 2c; \ b = 3c; \ d = 4c$

Possible schedules:
BBBCDDDDAA
BDBDBCADDA
BBDDDBDCAA
…
Dataflow process examples

- Design of a first 802.11b WLAN card
  - New product – combines RF, MAC and PHY; lots of DSP
  - Implemented on an ASIC
  - Previous design hand coded VHDL

- System level design with COSSAP by Synopsys
  - Explore architectural trade-offs – what in gates, what on DSP
  - Scheduling trade-offs: latency, area, propagation delay between clocks, and vendor library

- Results:
  - Reduced time to market by a factor of TWO!
  - Architectural exploration within 10% of actual measurements in terms of timing and area
Models of Computation

- State Machine Models
  - FSM, StateCharts, SDL, CFSM
- Petri nets
- Communicating Processes
  - Kahn processes, Communicating Sequential Processes
- Ada
- Dataflow models
  - DFG, SDFG
- Synchronous reactive languages
  - Cycle based models, Esterel, Lustre
- Discrete Event Systems
  - VHDL, Verilog, SystemC, SpecC
- HW Models
- Unified Modeling Language (UML)
Reactive Synchronous Languages

- Assumptions
  - Instantaneous reactions
  - Discrete event
  - Static

- Cycle based models
  - Excellent for (single) clocked synchronous circuits

- Control flow oriented (imperative) languages
  - Esterel

- Data flow languages
  - Lustre, Signal

- Deterministic behavior

- Simulation, software and hardware synthesis, verification
Lustre example
Reactive Synchronous Models: Esterel Statements

- Emit S
- Present S then p else q end
- Pause
- P; Q, P||Q
- Loop p end
- Await S
- Abort p when S
- Suspend p when S
- Sustain S = (loop emit S; pause end)
Abort Statement

```typescript
abort
    pause;
    pause;
    emit A
when B;
emit C
```

Normal Termination

Aborted termination

Aborted termination; emit A preempted

Normal Termination
B not checked in first cycle (like await)
Esterel Examples

emit A;
emit B;
pause;
loop
  present C then emit D end;
  present E then emit F end;
pause;
end

\[
\begin{array}{ccc}
A & D & D \\
B & F & F \\
C & C & E \\
E & E & E
\end{array}
\]
Esterel Examples

```
[ 
    await A; emit C 
||
    await B; emit D 
];
emit E
```

A \quad B

C \quad D \quad E
Time in Esterel

- Global clock with precise control over when events appear
  - At every tick: read inputs, compute, output

- Statements
  - A bounded number in one cycle
    - Emit, present, loop
  - Take multiple cycles:
    - Pause, await, sustain

- Causality analysis
  - Deterministic & non-contradictory
Esterel Application Examples

- TI used Esterel to automatically synthesize full coverage tests for a safety-critical design
  - Showed functional coverage covered only 30% of the design, with Esterl 100% covered

- Airbus
  - Significant decrease in errors due to increase in automated code generation (40-70%)
    - e.g. fly by wire controls & automatic flight control 70%, display computer 50%, warning & maintenance computer 40%
  - Major increase in productivity

<table>
<thead>
<tr>
<th>Computer Aided Specifications</th>
<th>A310 (70')</th>
<th>A320 (80')</th>
<th>A340 (90')</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aircraft</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number of digital units</td>
<td>77</td>
<td>102</td>
<td>115</td>
</tr>
<tr>
<td>Volume of on-board software in MB</td>
<td>10</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>Errors found per 100 KB</td>
<td>A few hundred</td>
<td>A few dozen</td>
<td>Less than 10</td>
</tr>
</tbody>
</table>
Esterel Summary

- Reactive synchronous language
- Control flow oriented
- Imperative syntax
- Synchrony assumption useful for safety critical embedded systems
  - Convert timing relations to causal ordering
  - Used in verification
    - e.g. TI, Airbus
Models of Computation

- State Machine Models
  - FSM, StateCharts, SDL, CFSM
- Petri nets
- Communicating Processes
  - Kahn processes, Communicating Sequential Processes
- Ada
- Dataflow models
  - DFG, SDFG
- Synchronous languages
  - Cycle based models, Esterel, Lustre
- Discrete Event Systems
  - VHDL, Verilog, SystemC, SpecC
- HW Models
- Unified Modeling Language (UML)
Discrete Events

- Notion of time is fundamental: global order
  - events are objects which carry ordered time info
  - there is a casual relationship between events
- DE simulator maintains global event queue
  - Verilog, VHDL
- Expensive - ordering tame stamps can be time consuming
- Large state & Low Activity => Effective simulation
- Simultaneous events lead to non-determinacy
  - require complex conflict resolution schemes
  - e.g. delta delays
Simultaneous Events in the Discrete Event Model

B has 0 delay

B has delta delay
**VHDL: Entities**

```vhdl
entity full_adder is
  port(a, b, carry_in: in Bit; -- input ports
       sum, carry_out: out Bit); --output ports
end full_adder;
```

![Diagram of full adder entity](image_url)
VHDL: Architectures

**architecture** structure of full_adder is

  **component** half_adder
    **port** (in1, in2: in Bit; carry, sum: **out** Bit);
  **end** component;

  **component** or_gate
    **port** (in1, in2: in Bit; o: **out** Bit);
  **end** component;

  **signal** x, y, z: Bit;  -- local signals

  **begin**  -- port map section
    i1: half_adder **port map** (a, b, x, y);
    i2: half_adder **port map** (y, carry_in, z, sum);
    i3: or_gate **port map** (x, z, carry_out);
  **end** structure;

**architecture** behavior of full_adder is

  **begin**
    sum <= (a xor b) xor carry_in **after** 10 Ns;
    carry_out <= (a and b) or (a and carry_in) or
                (b and carry_in) **after** 10 Ns;
  **end** behavior;
VHDL: signal strengths

\[ \begin{align*}
\text{'X'} & \quad \{ \text{strongest} \\
\text{'0'} & \quad \text{medium strength} \\
\text{'W'} & \quad \{ \text{pre-charged} \\
\text{'L'} & \quad \text{weakest} \\
\text{'H'} & \\
\text{'W'} & \\
\text{'l'} & \\
\text{'h'} & \\
\text{'Z'} &
\end{align*} \]
VHDL processes & waits

Processes model HW parallelism

```vhdl
process
begin
  a <= b after 10 ns
end
```

Waits:

```vhdl
wait until signal list;
wait until a;
wait until condition;
  wait until c='1';
wait for duration;
  wait for 10 ns;
wait; suspend indefinitely
wait on = sensitivity list
```

```vhdl
process (x, y)
begin
  prod <= x and y;
  wait on x,y;
end process;
```

```vhdl
process
begin
  prod <= x and y;
  wait for 10 ns;
  wait; suspend indefinitely
  wait on = sensitivity list
end process;
```
VHDL Simulation

1. Start of simulation
2. Future values for signal drivers
3. Assign new values to signals
4. Evaluate processes
5. Activate all processes sensitive to signal changes
VHDL Simulation of an RS FF

architecture one of RS_Flipflop is
begin
process: (R,S,Q,nQ)
begin
Q  <= R nor nQ;
nQ <= S nor Q;
end process;
end one;

δ cycles reflect the fact that no real gate comes with zero delay. Should delay-less signal assignments be allowed at all?
VHDL Summary

- Entities and architectures
- Multiple-valued logic
- Modeling hardware parallelism by processes
  - Wait statements and sensitivity lists
- VHDL simulation cycle
  - $\delta$ cycles, deterministic simulation
SystemC: Motivation
SystemC: Methodology
Models of Computation

- State Machine Models
  - FSM, StateCharts, SDL, CFSM
- Petri nets
- Communicating Processes
  - Kahn processes, Communicating Sequential Processes
- Ada
- Dataflow models
  - DFG, SDFG
- Synchronous languages
  - Cycle based models, Esterel, Lustre
- Discrete Event Systems
  - VHDL, Verilog, SystemC, SpecC
- HW Models
- Unified Modeling Language (UML)
Levels of hardware modeling

1. System level
2. Algorithmic level
3. Instruction set level
4. Register-transfer level (RTL)
5. Gate-level models
6. Switch-level models
7. Circuit-level models
8. Device-level models
9. Layout models
10. Process and device models
### Instruction level

<table>
<thead>
<tr>
<th>Assembler (MIPS)</th>
<th>Simulated semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>and $1,$2,$3</td>
<td>Reg[1]:=Reg[2] ∧ Reg[3]</td>
</tr>
<tr>
<td>or $1,$2,$3</td>
<td>Reg[1]:=Reg[2] ∨ Reg[3]</td>
</tr>
<tr>
<td>andi $1,$2,100</td>
<td>Reg[1]:=Reg[2] ∧ 100</td>
</tr>
<tr>
<td>sll $1,$2,10</td>
<td>Reg[1]:=Reg[2] &lt;&lt; 10</td>
</tr>
<tr>
<td>srl $1,$2,10</td>
<td>Reg[1]:=Reg[2] &gt;&gt; 10</td>
</tr>
</tbody>
</table>
Gate-level model
Switch level model
Circuit level model

.SUBCKT NAND2 VDD VSS A B OUT
MN1 I1 A VSS VSS NFET W=8U L=4U AD=64P AS=64P
MN2 OUT B I1 VSS NFET W=8U L=4U AD=64P AS=64P
MP1 OUT A VDD VDD PFET W=16U L=4U AD=128P AS=128P
MP2 OUT B VDD VDD PFET W=16U L=4U AD=128P AS=128P
CA A VSS 50fF
CB B VSS 50fF
COUT OUT VSS 100fF
.ENDS
Device level

Measured and simulated currents
Layout model

- powlo
- powhi
- dout
- din
Process model

N/[cm^3]

Simulated

Measured

X [μm]
Models of Computation

- State Machine Models
  - FSM, StateCharts, SDL, CFSM
- Petri nets
- Ada
- Communicating Processes
  - Kahn processes, Communicating Sequential Processes
- Dataflow models
  - DFG, SDFG
- Synchronous languages
  - Cycle based models, Esterel, Lustre
- Discrete Event Systems
  - VHDL, Verilog, SystemC, SpecC
- HW Models
- Unified Modeling Language (UML)
UML (Unified modeling language)
UML - StateCharts
UML – Extended Petri Nets
UML Summary

- UML
  - State machine diagram (StateChart-like)
  - Activity diagram (extended Petri nets)
  - Deployment diagram (exec. arch.)
  - Use case diagram
  - Package diagram (hierarchy)
  - Class diagrams
  - Timing diagrams (UML 2.0), UML for real-time
Models and Languages Summary

- Multiple models and languages are essential for high-level design
  - Managing complexity by abstraction
  - Formality ensures refinement correctness
  - Model choice depends on
    - Class of applications
    - Required operations (synthesis, scheduling, ...)

- Multiple MOCs can co-exist during all phases of design
  - Specification
  - Architectural mapping and simulation
  - Synthesis, code generation, scheduling
  - Detailed design and implementation
  - Co-simulation
Sources and References

- Alberto Sangiovanni-Vincentelli @ UCB
- Mani Srivastava @ UCLA
- Rajesh Gupta @ UCSD
- Nikil Dutt @ UCI