CSE 237A
Modeling Embedded Systems

Tajana Simunic Rosing
Department of Computer Science and Engineering
University of California, San Diego.
Until now...

- What are embedded systems
- State of industry
- Basic characteristics
  - Efficient, dependable, real-time, reactive, heterogeneous, UI...
- Examples of real ES
- Design example (PalmPilot)
- Challenges of HW/SW design of ES
ES Design

Hardware components

Verification and Validation
Models, Languages and Tools

Model M₁ → Language L₁ → Tool → Language L₂ → Model M₂

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

---

- Pascal
- C/C++
- Java
- VHDL

---

- Implementation A
- Implementation B
- Implementation C
Properties of languages

- Synchronous vs. asynchronous languages

- Properties of processes
  - Static vs. dynamic
  - Nesting
  - Process creation
Properties of languages: 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
}
```
Properties of languages: Timing

- **Measure elapsed time**

- **Delaying processes**
Properties of languages: Timing

- **Timeouts**

  - 20 ms
  - timeout

- **Deadlines**

  - execute
  - t
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
“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;

```c
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.*
Classical automata

- Moore-automata: 
  \[ O = H(S); \quad S^+ = f(I, S) \]
- Mealy-automata
  \[ O = H(I, S); \quad S^+ = f(I, S) \]
Finite-state machine (FSM) model

UnitControl process using a state machine

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

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

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

- DoorOpen
  - req < floor
  - !(req < floor)
  - u, d, o, t = 0, 0, 1, 1

- Timer:
  - timer < 10
  - u is up, d is down, o is open
  - t is timer_start
Template for FSM implementation in sequential programming

```c
#define S0 0
#define S1 1
...
#define SN N

void StateMachine() {
    int state = S0; // or whatever is the initial state.
    while (1) {
        switch (state) {
            case S0:
                // Insert S0’s actions here & Insert transitions T_i leaving S0:
                if (T_0’s condition is true) {state = T_0’s next state; /*actions*/}
                if (T_1’s condition is true) {state = T_1’s next state; /*actions*/}
                ...
                if (T_m’s condition is true) {state = T_m’s next state; /*actions*/}
                break;
            case S1:
                // Insert S1’s actions here
                // Insert transitions T_i leaving S1
                break;
            ...
            case SN:
                // Insert SN’s actions here
                // Insert transitions T_i leaving SN
                break;
        }
    }
}```
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
  - non-zero reaction time implies storage capability
Network of CFSMs

- GALS model, no hierarchy
UnitControl with FireMode

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

ElevatorController

UnitControl
- NormalMode
- FireMode

RequestResolver

!fire
fire

FireMode

UnitControl

RequestResolver

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

same meaning

Tajana Simunic Rosing
StateCharts: Concurrency

![Statechart Diagram]

- **answering-machine**
  - on
    - line-monitoring
      - ring
      - Lwait
      - Lproc
      - hangup (caller)
    - key-monitoring (excl. on/off)
      - key pressed
      - Kwait
      - Kproc
      - done

- key-on
- key-off
- off
Conditional Transitions
StateCharts: Example

- **Function & control behaviors**
  - **Function**: primary service of the entity
  - **Control**: actions performed within the system context
The Combined State Machine in StateChart Formalism

Uninitialized

reset/

Initialized

data/

Error

reset/

data/

stop/

start/

Operational

error/

ReadyToSendA

send/^A

SendingA

ackA/

ReadyToSendB

send/^B

SendingB

ackB/
StateCharts: Timers

```
\begin{center}
\begin{tikzpicture}
\node[shape=rectangle,draw,minimum size=4cm] (start) at (0,0) {};
\node[shape=circle,fill=white,draw,minimum size=2cm] (a) at ([shift={(-1,0)}]start) {};
\path[->] (a) edge[bend right] node[above] {20 ms} (start);
\path[->] (a) edge[bend right] node[above] {timeout} (start);
\end{tikzpicture}
\end{center}
```
Example: Answering machine

Lproc

4 s

timeout

play text

beep

8 s record

talk

return (callee)

dead

lift off

timeout

beep

timeout

silent
StateCharts: edge labels

event [condition] / reaction

swap

\[ e/a := b \]

\[ e/b := a \]

\[ a := 1; b := 0 \]
Propagations and Broadcasts

Source: B. P. Douglass & iLogix
StateCharts: Simulation

Status = values of all variables + set of events + current time
Step = execution of the three phases
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
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
SDL - Specification and Description Language
representation of FSMs/processes

Process P1

State: A, B, C, D, E
Input: g, h, i, j, f, k
Output: w, x, y, z, v, A
SDL - Operations on data

DCL
Counter Integer;
Date String;

Counter := Counter + 3;

Counter

(1:10) (11:30) ELSE
SDL - Communication among FSMs
SDL - Process interaction diagrams
SDL - Designation of recipients

Counter TO OFFSPRING

Counter Via Sw1

process P1 [A,B] Sw1 process P2

Sw2 [A]
SDL - Hierarchy

Block B

System S

Tajana Simunic Rosing
SDL - Timers

Process S

A → B → C → D → E

A → g → h → w → B

A → i → y → E

A → set(now+p,T) → E

E → f → T

E → v → RESET(T)

E → RESET(T)

E → A
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-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
Petri Nets

Preconditions

- train entering track from the left
- train leaving track to the right
- train wanting to go right
- train going to the right
- track available
- train going to the left
- single-laned

Tajana Simunic Rosing
Playing the „token game“

train wanting to go right

track available

train going to the right

train going to the left
Concurrency, Causality, Choice
Concurrency, Causality, Choice

Concurrency

Concurrency

Concurrency
Concurrency, Causality, Choice
Concurrenty, Causality, Choice

[Diagram of a Petri net with transitions t1, t2, t3, t4, t5, and t6, showing concurrency, causality, and choice conflicts.]
Concurrency, Causality, Choice
Conflict for resource „track“

- train wanting to go right
- train going to the right
- track available
- train going to the left
Condition/event nets

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
Communication Protocol
Producer-Consumer Problem
Producer-Consumer Problem
Producer-Consumer Problem
Producer-Consumer Problem
Producer-Consumer Problem

![Diagram of the Producer-Consumer Problem]
Producer-Consumer Problem
Producer-Consumer Problem
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

Consumer B can consume only if buffer A is empty.

Inhibitor arcs.
Until now…

- HW #1 due
- Topics:
  - Models of Computation vs. Languages
  - FSMs/CFSMs
  - StateCharts
  - SDL - Specification and Description Language
- Next time:
  - Computing platforms
  - Power management, voltage scaling
  - Mini project is out
StateCharts/SDL overview

− StateCharts
  − Hierarchy
    − AND- and OR-super states
    − Default state, History
  − Timing behavior
  − Broadcast, shared memory
  − Simulation – 3 phases for deterministic behavior

− Specification Design Language (SDL)
  − Representation of processes
  − Asynchronous comm.: unbounded FIFO
  − Timers – soft timing
  − Great for network protocols
Models of Computation

- State Machine Models
  - FSM, StateCharts, SDL, CFM
- 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
- HW Models
- Unified Modeling Language (UML)
Petri Nets

Preconditions

- Train entering track from the left
- Train leaving track to the right
- Train wanting to go right
- Train going to the right
- Track available
- Train going to the left
- Single-laned
Producer-Consumer with Priority

- Modeling synchronization control when sharing resources
  - Multiprocessor CPUs, distributed systems
Petri Net Properties

- **Behavioral**
  - Reachability
  - Boundedness
  - Schedulability
  - Liveness
  - Conservation

- **Structural**
  - Consistency
  - Structural boundedness
PN Properties - Reachability

\[
\begin{align*}
M_0 &= (1,0,1,0) \\
M &= (1,1,0,0)
\end{align*}
\]
PN Properties - Liveness

Not live
PN Properties - Liveness
PN Properties - Liveness

Deadlock-free
PN Properties - Liveness

Deadlock-free
PN Properties - Boundedness
PN Properties - Boundedness

Unbounded
PN Properties - Boundedness
PN Properties - Boundedness
PN Properties - Boundedness
PN Properties - Conservation

Not conservative
PN Properties - Conservation

Not conservative
PN Properties - Conservation

Conservative
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
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)
CSE 237A
Modeling Embedded Systems

Tajana Simunic Rosing
Department of Computer Science and Engineering
University of California, San Diego.
Last time...

- Topics:
  - Platforms
  - Power management
- Mini project out

Coming up:

- HW#2 out Thursday, due a week later
- Final project topic/teams due Tuesday, Jan 29th
  - Take a look at Fall’05 and Fall’06 cse237a website for sample projects
- Topics:
  - Models of computation – finish
  - Hardware – CPUs, Memory etc.
Mini Project

- **Goal:**
  - Design an energy efficient media player
    - audio playback needs to run in real time while saving maximum energy
    - General implementation
      - how well will your power manager adapt to different types of media files or other software running

- **Prizes for the top two implementations**
  - Given in class on Feb 12th
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
Petri Nets Example
Models of Computation

- State Machine Models
  - FSM, StateCharts, SDL, CFM
- 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, typically an infinite loop
  - Executes concurrently with other processes
- Basic operations on processes
  - Create and terminate
  - Suspend and resume
  - Join
- Process communication
  - Shared memory
  - Shared memory and mutexes
  - Message passing
- Properties
  - Continuous
  - Monothetic
Process networks

Alternating read of the two sensors

```
loop
    read_temp; read_humi
until false;
```
Dependences between processes/tasks

- main
- Get_temperature
- Get_humidity

FIFO
Task graphs

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

Arrival time  deadline

(0,7]  
\( T_1 \)  

(1,8]  
\( T_2 \)  

(3,10]  
\( T_3 \)
Task graphs - I/O

Diagram showing task graph connections with tasks labeled T1, T2, T3, T4, and T5.
Task graphs - Shared resources

- Task graphs
- Shared resources

Diagram:

- $T_1$ connected to $T_2$ and $T_3$
- $T_2$ connected to $T_3$ and $T_4$
- $T_3$ connected to $T_4$ and $T_5$
- $T_4$ connected to $T_5$

Tajana Simunic Rosing
Task graphs - Periodic schedules

\[ \ldots \rightarrow J_{n-1} \rightarrow J_n \rightarrow J_{n+1} \rightarrow \ldots \]

\[ \ldots \text{infinite task graphs} \]
Task graphs - Hierarchy
Concurrent processes

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
Concurrent process model: implementation

Diagram (a):
- Process 1
- Process 2
- Process 3
- Process 4

Diagram (b):
- Process 1
- Process 2
- Process 3
- Process 4

Diagram (c):
- Process 1
- Process 2
- Process 3
- Process 4

Processor A
Processor B
Processor C
Processor D

General Purpose Processor

Communication Bus
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;
Kahn process network

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

channel
process
Petri net model of a Kahn process

- KPNs are deterministic:
  - Process output determined by
    - Process, network, initial tokens
    - NOT by timing
      - Untimed model
Models of Computation

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

Control/Data Flow Graph

Implementation

Reg

Multiplier

Reg

Adder

0 4 7 ...

D

+ x

2 3 2 ...

2 1 1 ...

4 3 2 ...

4 7 9 ...

Tajana Simunic Rosing
Dataflow Process Networks

- **Dataflow:**
  - Maps input tokens to output tokens
  - Outputs function of only current inputs, not previous inputs
    - No need to keep state on suspend/resume
  - Scheduling needed to share resources

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

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.
Synchronous dataflow

Diagram showing a synchronous dataflow with nodes A and B connected by arrows indicating the flow of data.
SDF notation

- Nodes may have rates at which data are produced or consumed.
- Edges may have delays.
SDF example

- This graph has consistent sample rates:

![Graph with consistent sample rates]

separate outputs
Delays in SDF graphs

- Delays do not change rates, only the amount of data stored in the system.
- Changes system start-up.
Synchronous Data Flow

Diagram showing a network of operations including addition (+), subtraction (-), and multiplication (X) with feedback loops labeled D.
What Latency & Sample Period can a SDFG achieve?

- $T_L = \max(\text{I-O path, D-O path})$
- $T_S = \max(\text{D-D path, I-D path})$
SDFG can be Transformed to Affect $T_L$ & $T_S$
CSE 237A
Modeling Embedded Systems

Tajana Simunic Rosing
Department of Computer Science and Engineering
University of California, San Diego.
Last time…

- Topics:
  - Communicating processes: Ada, Kahn process networks, Dataflow networks, SDFs
- HW#2 out

Coming up:

- Final project topic/teams due Tuesday, Jan 29th
  - Take a look at Fall’05 and Fall’06 cse237a website for sample projects
- Topics:
  - SDF examples
  - Reactive synchronous languages
  - Finish MOCs
SDF examples

\[ \text{Diagram 1:} \quad n_1 \xrightarrow{a_1} n_2 \xrightarrow{a_2} n_3 \xrightarrow{a_3} n_1 \]

\[ \text{Diagram 2:} \quad n_1 \xrightarrow{a_1} n_2 \xrightarrow{a_2} n_3 \xrightarrow{a_3} n_1 \]
By building a set of “flow and conservation” equations

\[
\begin{align*}
3a - 2b &= 0 \\
4b - 3d &= 0 \\
b - 3c &= 0 \\
2c - a &= 0 \\
d - 2a &= 0
\end{align*}
\]

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

Possible schedules:
BBBCCDDDDDA
BBBDBBCADDA
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
- Cycle based models
  - Excellent for (single) clocked synchronous circuits
- Control flow oriented (imperative) languages
  - Esterel
- Data flow languages
  - Lustre, Signal
- Simple and clean semantics
  - based on FSMs.
- Deterministic behavior
- Simulation, software and hardware synthesis, verification
Simultaneous Events in the Discrete Event Model

B has 0 delay

B has delta delay
Lustre example

U -> pre -> - -> + -> sin -> * -> X

0.

pre

1.

pre

+ -> cos -> S
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
- Suspect p when S
- Sustain S = (loop emit S; pause end)
Abort Statement

```plaintext
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)
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
Esterel Examples

```plaintext
emit A;
emit B;
pause;
loop
  present C then emit D end;
  present E then emit F end;
pause;
end
```
Esterel Examples

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

A B

C D E
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 Esterel 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</td>
<td>4</td>
<td>10</td>
<td>20</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
VHDL: Entities

**entity** full_adder **is**

**port** (a, b, carry_in: **in** Bit; -- input ports
    sum, carry_out: **out** Bit); -- output ports

**end** full_adder;
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

- 'X': strongest
- '0', '1': medium strength
- 'W', 'H': pre-charged
- 'L', 'H': weakest
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
begin
  prod <= x and y;
  wait on x,y;
end process;
```

```vhdl
process (x, y)
begin
  prod <= x and y;
end process;
```
VHDL Simulation

Start of simulation

Future values for signal drivers

Assign new values to signals
Evaluate processes

Activate all processes sensitive to signal changes
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;

\( \delta \) cycles reflect the fact that no real gate comes with zero delay.

\( \mathcal{F} \) 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 sensivity 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><code>or $1,$2,$3</code></td>
<td><code>Reg[1] := Reg[2] \lor Reg[3]</code></td>
</tr>
<tr>
<td><code>andi $1,$2,100</code></td>
<td><code>Reg[1] := Reg[2] \land 100</code></td>
</tr>
<tr>
<td><code>sll $1,$2,10</code></td>
<td><code>Reg[1] := Reg[2] \ll 10</code></td>
</tr>
<tr>
<td><code>srl $1,$2,10</code></td>
<td><code>Reg[1] := Reg[2] \ll 10</code></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
Process model

![Graph showing simulated and measured N/cm³ against X [μm].]
Models of Computation

- State Machine Models
  - FSM, StateCharts, SDL, CFSM
- 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 nets
- HW Models
- Unified Modeling Language (UML)
UML (Unified modeling language)
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
Going Across MOC: Ptolemy

- Inter-domain simulations through domain encapsulation
  - Define semantics of every such encapsulation carefully, conservatively (and yet with some efficiency)
- Event horizon: Couple timed & untimed domains
How to cope with language problems in practice?

(RT-) UML or equivalent

SDL

C-programs

Assembly programs

Objectcode

(VHDL

Net list

hardware

(RT-) UML or equivalent

(RT-) Java

Objectcode
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