Introduction to Discrete Event Simulation

John Colley

12\textsuperscript{th} December 2011

Southampton
Introduction

• Motivation
• Discrete Event Simulation Principles
• Single-thread/Multi-thread Implementations
• Continuous Simulation Principles
• Hybrid Continuous/Discrete Simulation
• Summary
Discrete Event Simulation: Motivation

• Dominant Verification Technology for Synchronous Digital Hardware and Systems on Chip
  – $300 million per year Market
  – Mature Tools (30 years of development)
  – High Price Tools ($30k per seat + maintenance)

• Single Kernel, Multi-Language Tools for Model Re-use
  – VHDL
  – SystemVerilog
  – SystemC (C++)
  – C
Discrete Event Simulation

COMPONENT VIEW

Components: A, B, C, D (processes)
Connections: C1, C2 (unidirectional)
Ports: IN □ OUT ■

SIMULATOR API

GetValue(port)
HasChanged(port)
SetValue(OUT port, val, delay)
ScheduleEval(component, delay)
The Two-list Simulation Algorithm

t = 0

update list

t = n

evaluation list
The Two-list Simulation Algorithm

Why not just have a single, time-ordered list?
Time Zero Initialisation: Evaluate all Components

- A
- B
- C
- D

C1 connects A to B, and C2 connects C to D.

Update list:
- t = 0

Evaluation list:
- A
- B
- C
- D
Component evaluations call *SetValue, ScheduleEval*

A

B

C

D

C1

C2

t = 50  EC

t = 30  ED

t = 30  C2

t = 20  C1

t = 0

evaluation list

update list
Component evaluations call *Set*Value, *Schedule*Eval.

- **A**: C1 new val
- **B**: Schedule Eval, t=0
- **C**: future eval, t=30, update list
- **D**: future eval, t=30
- **C**: C2 new val, t=20
- **A**: C1 new val, t=20
- **C**: future eval, t=50

Evaluation list:
Global Time is Advanced to $t = 20$

- **Evaluation List**:
  - $t = 0$
  - $t = 20$ C1
  - $t = 30$ C2
  - $t = 30$ ED
  - $t = 50$ EC

- **Diagram**:
  - A to B via C1
  - C to D via C2
Add to *eval list* each component on C1 *fanout*
B calls *ScheduleEval* with delay 40

<table>
<thead>
<tr>
<th>Update list</th>
<th>Time (t)</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>C2</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>ED</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>EB</td>
<td>60</td>
<td></td>
</tr>
</tbody>
</table>

**Evaluation list**

- B
- D
- EB
- EC
- ED
- C2
- C1
Time advances to $t = 30$: two updates

- $t = 20$: C1
- $t = 30$: C2

Update list:
- $t = 20$: C1
- $t = 30$: C2, ED
- $t = 50$: EC
- $t = 60$: EB
Time advances to $t = 30$: *two* updates, *one* eval

- **C1** updates at $t = 30$
- **C2** updates at $t = 30$

**Update List:**
- $t = 60$: EB
- $t = 50$: EC
- $t = 30$: ED
- $t = 30$: C2

**Evaluation List:**
- D
Simple Algorithm, Complex Implementation for Performance
Simple Algorithm, Complex Implementation for Performance

Why not just have a single, time-ordered list?

RACE
Simulating Models without Discrete Delays – *Unit Delay*

Each evaluate/update cycle advances time by one *tick*
The Simulation Testbench

A

B

C

D

C1

C2

Primary inputs

Primary outputs
In *principle*, just another component using the API

In *practice*, a complex model of the design environment:
- Constrained Random Test Generation
- Assertion Checking
- Functional Coverage Metrics
Discrete Event Simulation Languages: Hierarchy
Discrete Event Simulation Languages: Hierarchy

Hierarchy is *flattened* for simulation
Discrete Event Simulation Languages: Function

• There must be an Entry Point to implement the \textit{Eval} API call

• \textit{Eval} call must execute to completion
  – Method call
  – Actor

• Languages with Embedded \textit{Wait}
  – \textit{Eval} call must resume at the Wait Point
  – Implement as Finite State Machine
    • Stored Current State represents the next Entry Point
A Concurrent Simulator Implementation?

![Diagram showing A, B, C, and D connected with C1 and C2, and an update list with t = n and t = 0.]
A Concurrent Simulator Implementation

Atomic Insertion

EVAL Components in Parallel

Update list

t = n
t = 0

Evaluation list
A Concurrent Simulator Implementation

Amdahl’s Law Revisited for Single Chip Systems
Jo-Ann M. Paul and Brett H. Meyer, 2006

Atomic Insertion
EVAL Components in Parallel

update list

evaluation list

$A_{\text{Concurrent Simulator Implementation}}$
Event-Driven vs Process-Driven

• Event-Driven
  – Nominally Single core, single process, \textit{BUT}
  – OpenMP implementation
    • Portable
    • Optimised for multi-core

• Process-Driven
  – Each process runs as a concurrent Thread, \textit{BUT}
    • Threads must manage communication/synchronisation

• Python Generators (coroutines)
  • Single core only (SimPy)
Continuous v Discrete Simulation

**Continuous**

- Equation-based
  
  model cont01
  
  Real x, y, z;
  
  equation
  
  x + y + z = 5.000;
  
  der(y) = x + 1.365;
  
  y = der(z) – 1.875;

  end cont01;

**Discrete**

- Assignment-based
  
  always @ (posedge clk)
  
  begin
  
  x <= x + 1
  
  y <= x - 1

  end
Hybrid Modeling

- Continuous-time + Discrete-time Modeling
  - Modelica “*when*”

```modelica
model t03
  Real x, y, z;
equation
  x + y + z = 5;
  when sample(0, 1) then
    x = x + 1;
    y = delay(x - 1, 20);
  end when;
end t03;
```
Summary

• Discrete Event Simulation
  – Two-List Algorithm for Deterministic Execution
  – “Event-driven” or “Process-driven”
  – Hardware/Software
  – Multi-thread implementation with OpenMP

• Continuous Simulation
  – Modelica, Matlab/Simulink

• Hybrid Continuous/Discrete Simulation
  • Discrete Interfaces