Discrete Event Models

Jian-Jia Chen
(slides are based on Peter Marwedel)
TU Dortmund, Informatik 12
Germany

These slides use Microsoft clip arts. Microsoft copyright restrictions apply.
Models of computation considered in this course

<table>
<thead>
<tr>
<th>Communication/local computations</th>
<th>Shared memory</th>
<th>Message passing</th>
<th>Synchronous</th>
<th>Asynchronous</th>
</tr>
</thead>
<tbody>
<tr>
<td>Undefined components</td>
<td>Plain text, use cases (Message) sequence charts</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Communicating finite state machines</td>
<td>StateCharts</td>
<td>SDL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data flow</td>
<td>Scoreboarding + Tomasulo Algorithm (Comp.Archict.)</td>
<td>Kahn networks, SDF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Petri nets</td>
<td>C/E nets, P/T nets, …</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Discrete event (DE) model</td>
<td>VHDL*, Verilog*, SystemC*, …</td>
<td>Only experimental systems, e.g. distributed DE in Ptolemy</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Von Neumann model</td>
<td>C, C++, Java</td>
<td>C, C++, Java with libraries CSP, ADA</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* Classification is based on implementation of VHDL, Verilog, SystemC with central queue
Discrete event semantics

Basic discrete event (DE) semantics

- Queue of future actions, sorted by time
- Loop:
  - Fetch next entry from queue
  - Perform function as listed in entry
    - May include generation of new entries
- Until termination criterion = true

<table>
<thead>
<tr>
<th>a</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>7</td>
</tr>
<tr>
<td>c</td>
<td>8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>queue</th>
<th>time</th>
<th>action</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>10</td>
<td>13</td>
</tr>
<tr>
<td>15</td>
<td>19</td>
<td>a:=5</td>
</tr>
<tr>
<td>6</td>
<td>a:=9</td>
<td>b:=7</td>
</tr>
<tr>
<td>c:=8</td>
<td>a:=6</td>
<td>b:=7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>a:=9</td>
</tr>
</tbody>
</table>
HDLs using discrete event (DE) semantics

Used in hardware description languages (HDLs):
Description of concurrency is a must for HW description languages!

- Many HW components are operating concurrently
- Typically mapped to “processes“
- These processes communicate via “signals“

Examples:
- MIMOLA [Zimmermann/Marwedel], ~1975 …
- …. 
- VHDL (very prominent example in DE modeling)

One of the 3 most important HDLs:
VHDL, Verilog, SystemC
VHDL

VHDL = VHSIC hardware description language
VHSIC = very high speed integrated circuit
1980: Def. started by US Dept. of Defense (DoD) in 1980
1984: first version of the language defined, based on ADA (which in turn is based on PASCAL)
1987: revised version became IEEE standard 1076
1992: revised IEEE standard
1999: VHDL-AMS: includes analog modeling
2006: Major extensions
Simple example (VHDL notation)

Processes will wait for changes on their input ports. If they arrive, processes will wake up, compute their code and deposit changes of output signals in the event queue and wait for the next event. If all processes wait, the next entry will be taken from the event queue.
VHDL processes

Delays allowed:

\[
\text{process } (a,b) \\
\begin{align*}
\text{begin} & \quad c \leq a \text{ nor } b \text{ after } 10 \text{ ns;} \\
\text{end;}
\end{align*}
\]

Equivalent to

\[
\text{process} \\
\begin{align*}
\text{begin} & \quad c \leq a \text{ nor } b \text{ after } 10 \text{ ns;} \\
\text{wait on} & \quad a,b; \\
\text{end;}
\end{align*}
\]

- \(\leq\): signal assignment operator
- Each executed signal assignment will result in \textit{adding} entries in the projected waveform, as indicated by the (optional) delay time
- Implicit loop around the code in the body
- Sensitivity lists are a shorthand for a single \textit{wait on}-statement at the end of the process body
The full adder as an example

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

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;
The full adder as an example
- Simulation results -
VHDL semantics: global control

According to the original VHDL standards document:

- The execution of a model consists of an initialization phase followed by the repetitive execution of process statements in the description of that model.

- Initialization phase executes each process once.
VHDL semantics: initialization

At the beginning of initialization, the current time, $T_c$ is 0 ns.

- The … effective value of each explicitly declared signal are computed, and the current value of the signal is set to the effective value. …

- Each … process … is executed until it suspends.

- The time of the next simulation cycle (… in this case … the 1st cycle), $T_n$ is calculated according to the rules of step f of the simulation cycle, below.
According to the standard, the simulation cycle is as follows:

a) Stop if $T_n = \text{time'high}$ and "nothing else is to be done" at $T_n$.

The current time, $T_c$ is set to $T_n$. 

![Diagram showing the simulation cycle]

- Activate all processes sensitive to signal changes
- Evaluate processes
- Assign new values to signals
- Future values for signal drivers
- EXIT

?
VHDL semantics: The simulation cycle (2)

a) *Each active explicit signal in the model is updated.* *(Events may occur as a result.)*
Previously computed entries in the queue are now assigned if their time corresponds to the current time $T_c$.
New values of signals are not assigned before the next simulation cycle, at the earliest.
Signal value changes result in events $\mathcal{E}$ enable the execution of processes that are sensitive to that signal.

b) ..
VHDL semantics: The simulation cycle (3)

a) ∀ P sensitive to s: if event on s in current cycle: P resumes.

b) Each ... process that has resumed in the current simulation cycle is executed until it suspends*. 
   *Generates future values for signal drivers.
VHDL semantics: The simulation cycle (4)

a) Time $T_n$ of the next simulation cycle = earliest of
   1. time ‘high (end of simulation time).
   2. The next time at which a driver becomes active
   3. The next time at which a process resumes
      (determined by `wait for` statements).

Next simulation cycle (if any) will be a delta cycle if $T_n = T_c$. 
δ-simulation cycles

Next simulation cycle (if any) will be a delta cycle if $T_n = T_c$. Delta cycles are generated for delay-less models. There is an arbitrary number of δ cycles between any 2 physical time instants:

In fact, simulation of delay-less hardware loops might not terminate (don’t even advance $T_c$).
δ-simulation cycles
Simulation of an RS-Flipflop

<table>
<thead>
<tr>
<th>0ns</th>
<th>0ns+δ</th>
<th>0ns+2δ</th>
<th>0ns+3δ</th>
</tr>
</thead>
<tbody>
<tr>
<td>R</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>S</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Q</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>nQ</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

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

gate1:
process (S,Q)
begin
nQ <= S nor Q;
end;
geate2:
process (R,nQ)
begin
Q <= R nor nQ;
end;

F
δ-simulation cycles
and determinate simulation semantics

Semantics of

a <= b;

b <= a;  ?

Separation into 2 simulation phases results in determinate semantics ((StateMate).