Discrete Event Models

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

2017年 10 月 25日
## Models of computation considered in this course

<table>
<thead>
<tr>
<th>Communication/local computations</th>
<th>Shared memory</th>
<th>Message passing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Undefined components</td>
<td>Plain text, use cases</td>
<td>Synchronous</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Communicating finite state machines</td>
<td>StateCharts</td>
<td>SDL</td>
</tr>
<tr>
<td>Data flow</td>
<td>Scoreboarding + Tomasulo Algorithm</td>
<td>Kahn networks, SDF</td>
</tr>
<tr>
<td></td>
<td>(Comp.Archict.)</td>
<td></td>
</tr>
<tr>
<td>Petri nets</td>
<td>C/E nets, P/T nets, …</td>
<td></td>
</tr>
<tr>
<td>Discrete event (DE) model</td>
<td>VHDL*, Verilog*, SystemC*, …</td>
<td>Only experimental systems, e.g.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>distributed DE in Ptolemy</td>
</tr>
<tr>
<td>Von Neumann model</td>
<td>C, C++, Java</td>
<td>C, C++, Java with libraries</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CSP, ADA</td>
</tr>
</tbody>
</table>
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>time</th>
<th>action</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>a:=5</td>
</tr>
<tr>
<td>10</td>
<td>b:=7</td>
</tr>
<tr>
<td>13</td>
<td>c:=8</td>
</tr>
<tr>
<td>15</td>
<td>a:=6</td>
</tr>
<tr>
<td>19</td>
<td>a:=9</td>
</tr>
</tbody>
</table>

Queue of future actions, sorted by time: a:=5, b:=7, c:=8, a:=6, a:=9
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:

```
process (a,b)
begin
  c <= a nor b after 10 ns;
end;
```

Equivalent to

```
process
begin
  c <= a nor b after 10 ns;
  wait on a,b;
end;
```

- `<=`: signal assignment operator
- Each executed signal assignment will result in 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 `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 $T_n = \text{time'high}$ or “nothing else is to be done” at $T_n$. Otherwise, the current time, $T_c$ is set to $T_n$. 

---

Exit

? Future values for signal drivers

Assign new values to signals Evaluate processes

Activate all processes sensitive to signal changes
b) *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 $F$ enable the execution of processes that are sensitive to that signal.

c)....
d) $\forall P$ sensitive to $s$: if event on $s$ in current cycle: $P$ resumes.
e) Each ... process that has resumed in the current simulation cycle is executed until it suspends*.

*Generates future values for signal drivers.
f) 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 $T_n = T_c$. 
\( \delta \)-simulation cycles

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

\[
\begin{align*}
T & \quad T+1 & \quad T+2 & \quad T+3 & \quad \ldots \\
\end{align*}
\]

\text{In fact, simulation of delay-less hardware loops might not terminate (don’t even advance } T_c).
What is the delta cycle of the following circuit?
δ-simulation cycles
Simulation of an RS-Flipflop

δ cycles reflect the fact that no real gate comes with zero delay.

Should delay-less signal assignments be allowed at all?
δ-simulation cycles and determinate simulation semantics

Semantics of

\[ a \leq b; \]
\[ b \leq a; \]

Separation into 2 simulation phases results in determinate semantics (StateMate).
Multi-valued logic and standard IEEE 1164

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

2017年 11 月 06 日

These slides use Microsoft clip arts. Microsoft copyright restrictions apply.
Abstraction of electrical signals

❖ We try to use digital values and DE simulation as long as possible
❖ However, using just 2 digital values would be too restrictive (as we will see)
2 signal strengths

Many subcircuits can effectively disconnect themselves from the rest of the circuit (they provide “high impedance” values to the rest of the circuit). Example: subcircuits with open collector or tri-state outputs.

\[ \text{Input} = '0' \rightarrow \text{A disconnected} \]
We introduce signal value 'Z'
We introduce an operation \( \# \), which generates the effective signal value whenever two signals are connected by a wire.

\[ #('0', 'Z') = '0'; #('1', 'Z') = '1'; '0' \text{ and } '1' \text{ are “stronger” than 'Z'} \]

In order to define \#('0', '1'), we introduce 'X', denoting an undefined signal level. 'X' has the same strength as '0' and '1'.

According to the partial order in the diagram, \# returns the smallest element at least as large as the two arguments (“Sup”).
Application example

signal value on bus = #(value from left subcircuit, value from right subcircuit)
#('Z', value from right subcircuit)
value from right subcircuit
"as if left circuit were not there".
3 signal strengths

There may also be weak signals of the same level as '0'

- Introduction of 'L':
  \[ \#('L', '1') = '1'; \quad \#('L', 'Z') = 'L'; \]

- Similarly for 'H':
  \[ \#('H', '1') = '1'; \quad \#('H', 'Z') = 'H'; \]

- Introduction of 'W':
  \[ \#('L', 'H') = 'W'; \quad \#('L', 'W') = 'W'; \]

# reflected by the partial order shown.
IEEE 1164

VHDL allows user-defined value sets.

Each model could use different value sets (unpractical)

Definition of standard value set according to standard IEEE 1164:

\{'0', '1', 'Z', 'X', 'H', 'L', 'W', 'U', '-'\}

First seven values as discussed previously.

Everything said about 7-valued logic applies.

'U': un-initialized signal; used by simulator for signals that have not been explicitly initialized.

'-': denotes input don't care.
Input don‘t care

'-' denotes input don't care.

Suppose:

\[ f(a, b, c) = a \bar{b} + bc \]

except for \(a=b=c='0'\) where \(f\) is undefined

Then, we could like specifying this in VHDL as

\[ f <= \text{select} \ a \ & \ b \ & \ c \]

\[ '1' \ \text{when} \ "10-" \quad \text{-- first term} \]

\[ '1' \ \text{when} \ "-11" \quad \text{-- second term} \]

\[ 'X' \ \text{when} \ "000" \quad \text{-- 'X' \(\triangleq\) ('0' or '1') here (output don't care)} \]

\[ '0' \ \text{otherwise}; \]

("input don‘t care -“ was with different behaviour before VHDL’2006).
Using # (=sup) in resolution functions in VHDL

```vhdl
constant resolution_table : stdlogic_table := (
  --U  X  0  1  Z  W  L  H  --
  ('U', 'U', 'U', 'U', 'U', 'U', 'U', 'U'),   --| U |
  ('U', 'X', 'X', 'X', 'X', 'X', 'X', 'X'),   --| X |
  ('U', 'X', '0', 'X', '0', '0', '0', 'X'),   --| 0 |
  ('U', 'X', '1', '1', '1', '1', '1', '1'),   --| 1 |
  ('U', 'X', '0', '1', 'Z', 'W', 'L', 'H', 'X'),   --| Z |
  ('U', 'X', '0', '1', 'W', 'W', 'W', 'H', 'X'),   --| W |
  ('U', 'X', '0', '1', 'L', 'W', 'L', 'W', 'X'),   --| L |
  ('U', 'X', '0', '1', 'H', 'W', 'W', 'H', 'X'),   --| H |
  ('U', 'X', 'X', 'X', 'X', 'X', 'X', 'X'),   --| - |
);

This table would be difficult to understand without the partial order
```

```
0' 1'
W'  L'  H'
Z'
```


Summary

Discrete event models

- Queue of future events, fetch and execute cycle, commonly used in HDLs
- processes model HW concurrency
- signals model communication
- \textit{wait}, sensitivity lists
- the VHDL simulation cycle
  \[ \forall \delta \text{ cycles, determinate simulation} \]

Multiple-valued logic

- General approach
- Application to IEEE 1164