Modeling of Time in Discrete-Event Simulation of Systems-on-Chip

Giovanni Funchal\textsuperscript{1,2} and Matthieu Moy\textsuperscript{1}

\textsuperscript{1}Verimag (Grenoble INP)  
Grenoble, France  

\textsuperscript{2}STMicroelectronics  
Grenoble, France  

Work partially supported by  
HELP ANR project

MEMOCODE, July 2011
Outline

1. Transaction Level Modeling and jTLM
2. Time and Duration in jTLM
3. Applications
4. Implementation
5. Conclusion
Outline

1. Transaction Level Modeling and jTLM
2. Time and Duration in jTLM
3. Applications
4. Implementation
5. Conclusion
Modern Systems-on-a-Chip
Modern Systems-on-a-Chip

Software

Hardware
Transaction-Level Modeling

(Fast) simulation essential in the design-flow
- To write/debug software
- To validate architectural choices
- As reference for hardware verification
Transaction-Level Modeling

(Fast) simulation essential in the design-flow
- To write/debug software
- To validate architectural choices
- As reference for hardware verification

Transaction-Level Modeling (TLM):
- High level of abstraction
- Suitable for
Transaction-Level Modeling

- (Fast) simulation essential in the design-flow
  - To write/debug **software**
  - To validate **architectural choices**
  - As reference for hardware **verification**

- Transaction-Level Modeling (TLM):
  - High level of abstraction
  - Suitable for

  **Industry Standard** = SystemC/TLM
SystemC/TLM vs. “TLM Abstraction Level”
SystemC/TLM vs. “TLM Abstraction Level”

SystemC

- Cycle accurate
- RTL
- Gate level

TLM

- TLM 2.0

jTLM = this talk
SystemC/TLM vs. “TLM Abstraction Level”

- SystemC
  - Cycle accurate
  - RTL
  - Gate level

- TLM
  - TLM 2.0
  - ?
SystemC/TLM vs. “TLM Abstraction Level”

SystemC
- Cycle accurate
- Clocks
- RTL
- Coroutine semantics
- Gate level
- δ-cycle

TLM
- Parallelism
- Function calls
- TLM 2.0

?
SystemC/TLM vs. “TLM Abstraction Level”

SystemC

- Cycle accurate
- RTL
- Clocks
- Coroutine semantics
- Gate level $\delta$-cycle

TLM

- Parallelism
- Function calls

$jTLM = this talk$
jTLM: goals and peculiarities

- jTLM’s goal: define “TLM” independently of SystemC
  - Not cooperative (true parallelism)
  - Not C++ (Java)
  - No $\delta$-cycle

- Interesting features
  - Small and simple code ($\approx 500$ LOC)
  - Nice experimentation platform

- Not meant for production
Outline

1. Transaction Level Modeling and jTLM
2. Time and Duration in jTLM
3. Applications
4. Implementation
5. Conclusion
Simulation Time Vs Wall-Clock Time

Wall-clock time

Time elapse

Computation

Simulation time

0 10 20 30 40
Time in SystemC and jTLM

- Process A:
  ```
  // computation
  f();
  // time taken by f
  wait(20, SC_NS);
  ```

- Process P:
  ```
  g();
  awaitTime(20);
  consumesTime(15) {
    h();
  }
  ```

SystemC
A → B

jTLM
P → Q
Time in SystemC and jTLM

**Process A:**
```c
// computation
f();
// time taken by f
wait(20, SC_NS);
```

**SystemC**
- A → B

**jTLM**
- P → Q
Time in SystemC and jTLM

**Process A:**

```plaintext
// computation
f();
// time taken by f
wait(20, SC_NS);
```

### SystemC

Process A:
```
f()
wait(20)
```

### jTLM

```
P
Q
```
### Time in SystemC and jTLM

**Process A:**

```c
// computation
f();
// time taken by f
wait(20, SC_NS);
```

**Process P:**

```c
g();
awaitTime(20);
```
Time in SystemC and jTLM

**Process A:**
```cpp
// computation
f();
// time taken by f
wait(20, SC_NS);
```

**Process P:**
```cpp
g();
awaitTime(20);
consumesTime(15) {
  h();
}
```
Time in SystemC and jTLM

**Process A:**
```c
// computation
f();
// time taken by f
wait(20, SC_NS);
```

**Process P:**
```c
g();
awaitTime(20);
consumesTime(15) {
    h();
}
```
Time à la SystemC: \texttt{awaitTime}(T)

- By default, time does not pass \Rightarrow instantaneous tasks
- \texttt{awaitTime}(T): let other processes execute for \( T \) time units

```plaintext
f(); // instantaneous
awaitTime(20);
```
Task with Known Duration: `consumesTime(T)`

- **Semantics:**
  - Start and end dates known
  - Actions contained in task spread in between

- **Advantages:**
  - Model closer to actual system
  - Less bugs hidden
  - Better parallelization

```java
consumesTime(15) {
    f1();
    f2();
    f3();
}
consumesTime(10) {
    g();
}
```
Execution of $\text{consume} \text{Time}(T)$

- **Slow computation**
  - Simulation time blocked
  - Task starts
  - Task finishes

- **Fast computation**
  - Computation ends
  - Task finishes
  - Rest of the platform drives time
  - idle

Wall-clock time

Simulation time
Outline

1. Transaction Level Modeling and jTLM
2. Time and Duration in jTLM
3. Applications
4. Implementation
5. Conclusion
Exposing Bugs

Example bug: mis-placed synchronization:

```plaintext
flag = true;
awaitTime(5);
writeIMG();
awaitTime(10);

while(!flag)
awaitTime(1);
awaitTime(10);
readIMG();

⇒ bug never seen in simulation
```
Exposing Bugs

Example bug: mis-placed synchronization:

```java
flag = true;
awaitTime(5);
writeIMG();
awaitTime(10);
while(!flag)
    awaitTime(1);
awaitTime(10);
readIMG();
```

⇒ bug never seen in simulation

```java
consumesTime(15) {
    flag = true;
    writeIMG();
    awaitTime(10);
    readIMG();
}
```

⇒ strictly more behaviors, including the buggy one
Parallelization

- **jTLM’s Semantics**
  - Simultaneous tasks run in parallel
Parallelization

jTLM’s Semantics

- Simultaneous tasks run in parallel
- Non-simultaneous tasks don’t
Parallelization

jTLM’s Semantics

- Simultaneous tasks run in parallel
- Non-simultaneous tasks don’t
- Overlapping tasks do

Simultaneous tasks run in parallel
Non-simultaneous tasks don’t
Overlapping tasks do

Back to SystemC:

- ▶ Parallelizing within δ-cycle = great if you have clocks
- ▶ Simulation time is the bottleneck with quantitative/fuzzy time
Parallelization

jTLM’s Semantics
- Simultaneous tasks run in parallel
- Non-simultaneous tasks don’t
- Overlapping tasks do

Back to SystemC:
- Parallelizing within $\delta$-cycle = great if you have clocks
- Simulation time is the bottleneck with quantitative/fuzzy time
Outline

1. Transaction Level Modeling and jTLM
2. Time and Duration in jTLM
3. Applications
4. Implementation
5. Conclusion
Time Queue and **awaitTime**(T)

Current instant

Process P:
- \( f() \)
- `awaitTime(50)`

Process Q:
- \( h() \)
- `awaitTime(30)`
- `g()`
- `awaitTime(30)`

Process R:
- \( i() \)
- `awaitTime(90)`
**Time Queue and `awaitTime(T)`**

**Process P:**
- `f();`
- `awaitTime(50);`

**Process Q:**
- `h();`
- `awaitTime(30);`
- `g();`
- `awaitTime(30);`

**Process R:**
- `i();`
- `awaitTime(90);`

Current instant: `awaitTime(50)`

Matthieu Moy (Verimag)
**Time Queue and `awaitTime(T)`**

Current instant

`awaitTime(30)`

**Process P:**
```plaintext
f();
awaitTime(50);
```

**Process Q:**
```plaintext
h();
awaitTime(30);
g();
awaitTime(30);
```

**Process R:**
```plaintext
i();
awaitTime(90);
```
**Time Queue and `awaitTime(T)`**

**Current instant**

- **Process P:**
  
  ```
  f();
  ▶ `awaitTime(50)`;
  ```

- **Process Q:**
  
  ```
  h();
  ▶ `awaitTime(30)`;
  g();
  `awaitTime(30)`;
  ```

- **Process R:**
  
  ```
  i();
  ▶ `awaitTime(90)`;
  ```

**Diagram:**

- `awaitTime(90)`
- Current instant
- Processes P, Q, R
**Time Queue and `awaitTime(T)`**

**Process P:**
- `f();`
- `awaitTime(50);`

**Process Q:**
- `h();`
- `awaitTime(30);`
- `g();`
- `awaitTime(30);`

**Process R:**
- `i();`
- `awaitTime(90);`
Time Queue and \texttt{awaitTime(T)}

\begin{itemize}
\item \textbf{Process P:} \( \texttt{f();} \) \( \texttt{\triangleright \texttt{awaitTime(50);}} \) \\
\item \textbf{Process Q:} \( \texttt{\triangleright h();} \) \( \texttt{\triangleright \texttt{awaitTime(30);}} \) \( \texttt{\triangleright g();} \) \( \texttt{\triangleright \texttt{awaitTime(30);}} \) \\
\item \textbf{Process R:} \( \texttt{\triangleright i();} \) \( \texttt{\triangleright \texttt{awaitTime(90);}} \)
\end{itemize}
Time Queue and `awaitTime(T)`

**Process P:**
- `f();`
- `awaitTime(50);`

**Process Q:**
- `h();`
- `awaitTime(30);`
- `g();`
- `awaitTime(30);`

**Process R:**
- `i();`
- `awaitTime(90);`
- `awaitTime(30);`
Time Queue and $\text{consumesTime}(T)$

What about $\text{consumesTime}(T)$?
Time Queue and `consumesTime(T)`

Current instant

Process P:
- `f();
  consumesTime(50){
    g();
  }
  h();`

Process Q:
- `i();
  awaitTime(30);
  j();
  consumesTime(30){
    k();
  }`

Process R:
- `l();
  awaitTime(90);`
Time Queue and `consumesTime(T)`

**Process P:**
```
f();
consumesTime(50) {
  g();
}
h();
```

**Process Q:**
```
i();
awaitTime(30);
j();
consumesTime(30) {
  k();
}
```

**Process R:**
```
l();
awaitTime(90);
```
Time Queue and \texttt{consumesTime(T)}

**Process P:**
\begin{verbatim}
  f();
  consumesTime(50) {
    g();
  }
  h();
\end{verbatim}

**Process Q:**
\begin{verbatim}
  i();
  awaitTime(30);
  j();
  consumesTime(30) {
    k();
  }
\end{verbatim}

**Process R:**
\begin{verbatim}
  l();
  awaitTime(90);
\end{verbatim}
Time Queue and `consumesTime(T)`

Current instant

AwaitTime(30)

Process P:
```java
f();
consumesTime(50) {
   g();
}
h();
```

Process Q:
```java
i();
awaitTime(30);
j();
consumesTime(30) {
   k();
}
```

Process R:
```java
l();
awaitTime(90);
```
Time Queue and `consumesTime(T)`

**Current instant**

- **Process P:**
  ```java
  f();
  consumesTime(50) {
    g();
  }
  h();
  ```

- **Process Q:**
  ```java
  i();
  ▶ awaitTime(30);
  j();
  consumesTime(30) {
    k();
  }
  ```

- **Process R:**
  ```java
  l();
  ▶ awaitTime(90);
  ```

**Time Elapse**

- **Current instant**

**awaitTime(90)**
Time Queue and `consumesTime(T)`

Process P:
```java
f();
consumesTime(50){
  g();
}
h();
```

Process Q:
```java
i();
waitTime(30);
j();
consumesTime(30){
  k();
}
```

Process R:
```java
l();
waitTime(90);
```
Time Queue and `consumesTime(T)`

Process P:

```java
f();
consumesTime(50){
    g();
} 

h();
```

Process Q:

```java
i();
awaitTime(30);

j();
consumesTime(30){
    k();
}
```

Process R:

```java
l();
awaitTime(90);
```
Time Queue and `consumesTime(T)`

**Current instant**

```
Process P:
f();
consumesTime(50) {
  g();
}
h();
```

```
Process Q:
i();
awaitTime(30);
j();
consumesTime(30) {
  k();
}
```

```
Process R:
l();
awaitTime(90);
```

Matthieu Moy (Verimag)
Modeling of Time/jTLM
MEMOCODE, July 2011
**Time Queue and `consumesTime(T)`**

**Current instant**

**Process P:**
```
f();
consumesTime(50){
  g();
}
h();
```

**Process Q:**
```
i();
awaitTime(30);
j();
consumesTime(30){
  k();
}
```

**Process R:**
```
l();
▷awaitTime(90);
```
Time Queue and `consumesTime(T)`

Current instant

Time Elapse

Process P:

\[ f(); \]
\[ consumesTime(50) \{
  \quad g();
\} \]
\[ h(); \]

Process Q:

\[ i(); \]
\[ awaitTime(30); \]
\[ j(); \]
\[ consumesTime(30) \{
  \quad k();
\} \]

Process R:

\[ l(); \]
\[ \leftarrow awaitTime(90); \]
Time Queue and `consumesTime(T)`

**Process P:**

```plaintext
f();
consumesTime(50){
g();
}

h();
```

**Process Q:**

```plaintext
i();
awaitTime(30);
j();
consumesTime(30){

k();
}
```

**Process R:**

```plaintext
l();
\[\text{awaitTime}(90);\]
```
Outline

1. Transaction Level Modeling and jTLM
2. Time and Duration in jTLM
3. Applications
4. Implementation
5. Conclusion
Perspectives

Summary
- Tasks with duration
- Exhibit more behaviors/bugs
- Better parallelization

Skipped from the talk (cf. paper)
- Tasks with a priori unknown duration
- jTLM’s cooperative mode

Perspectives
- Adapt the ideas to SystemC (ongoing, not so hard)
- Run-time Verification to explore schedules (science-fiction)
- Open-Source Release?
Perspectives

Summary
- Tasks with duration
- Exhibit more behaviors/bugs
- Better parallelization

Skipped from the talk (cf. paper)
- Tasks with a priori unknown duration
- jTLM’s cooperative mode

Perspectives
- Adapt the ideas to SystemC (ongoing, not so hard)
- Run-time Verification to explore schedules (science-fiction)
- Open-Source Release?

Thank you! ~ Questions?