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

Giovanni Funchal[1,2] and Matthieu Moy[1]

[1] Verimag (Grenoble INP)
Grenoble, France

[2] STMicroelectronics
Grenoble, France

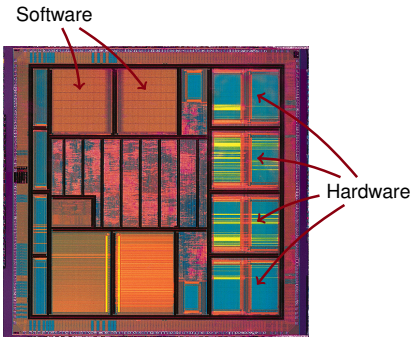
Work partially supported by
HELP ANR project

MEMOCODE, July 2011

Abstract

Outline

Modern Systems-on-a-Chip

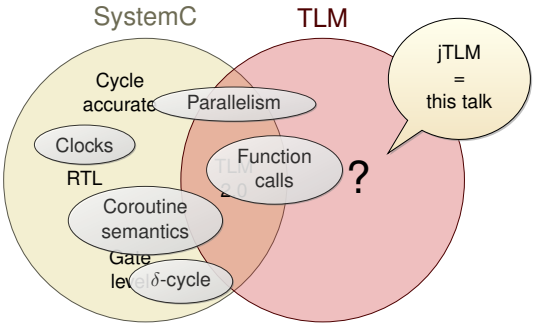


Transaction-Level Modeling

- (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

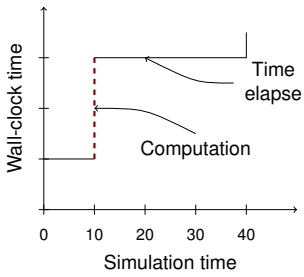
SystemC/TLM vs. “TLM Abstraction Level”



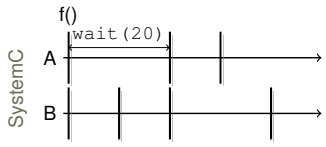
jTLM: goals and peculiarities

- jTLM’s goal: define “TLM” independently of SystemC
  - Not cooperative (true parallelism)
  - Not C++ (Java)
  - No δ-cycle
- Interesting features
  - Small and simple code ($\approx$ 500 LOC)
  - Nice experimentation platform
- Not meant for production

Simulation Time Vs Wall-Clock Time

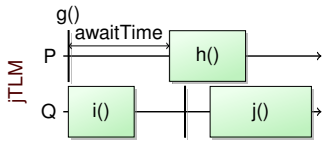


Time in SystemC and jTLM



Process A:

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

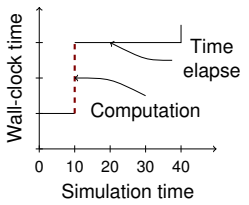


Process P:

```
g();
awaitTime(20);
consumesTime(15) {
  h();
}
```


Time *à la* SystemC: `awaitTime(T)`

- By default, time does not pass
  ⇒ instantaneous tasks
- `awaitTime(T)` :
  let other processes execute for T time units



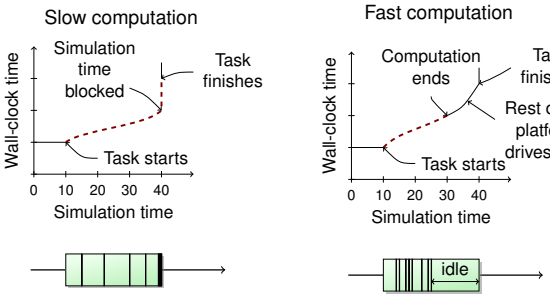

```
f(); // instantaneous
awaitTime(20);
```


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

```
consumesTime(15) {
    f1();
    f2();
    f3();
}
consumesTime(10) {
    g();
}
```


Execution of `consumesTime(T)`



Exposing Bugs

Example bug: mis-placed synchronization:

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

||

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

⇒ bug never seen in simulation

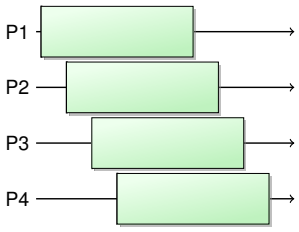
```
consumesTime(15) {
    flag = true;
    writeIMG();
}
```

||

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

⇒ strictly more behaviors, including the buggy one

Parallelization

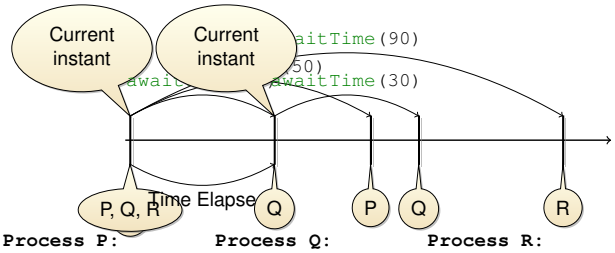


jTLM's Semantics

- 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

Time Queue and `awaitTime(T)`



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

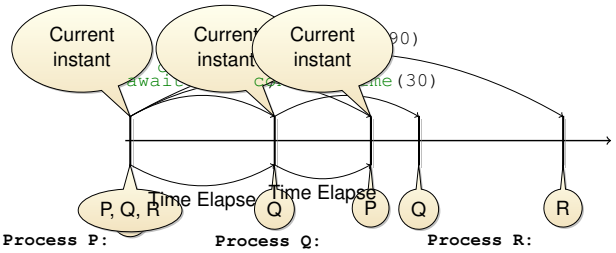
Process Q:
- `h();`
- `awaitTime(30);`
- `g();`
- `awaitTime(30);`

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

Time Queue and `consumesTime(T)`

What about `consumesTime(T)` ?

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);
```


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! $\rightsquigarrow$ Questions?