# Test Coverage for Loose Timing Annotations

Claude Helmstetter
and Florence Maraninchi
and Laurent Maillet-Contoz

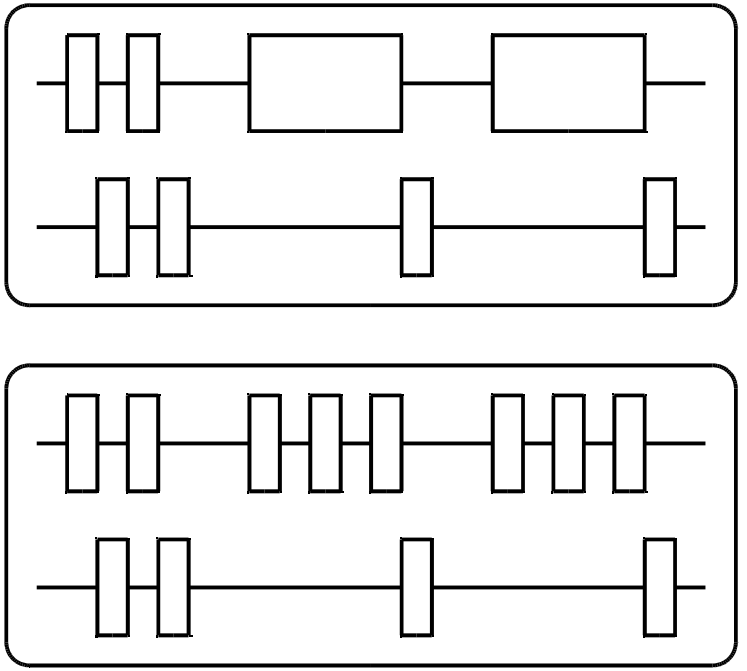
Verimag &
ST Microelectronics

# Context: Transaction Level Model

fastest

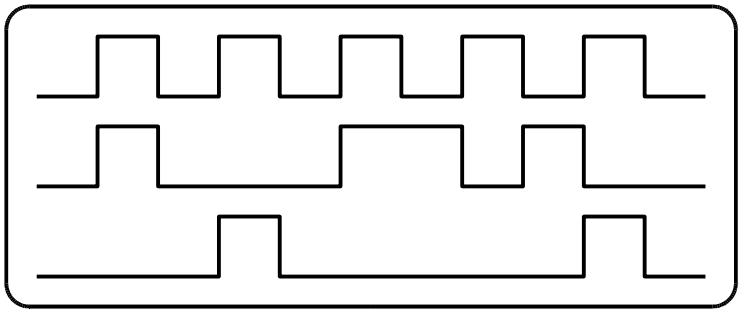
Early simulation
of the embedded
software

Golden model for
RTL validation



TLM

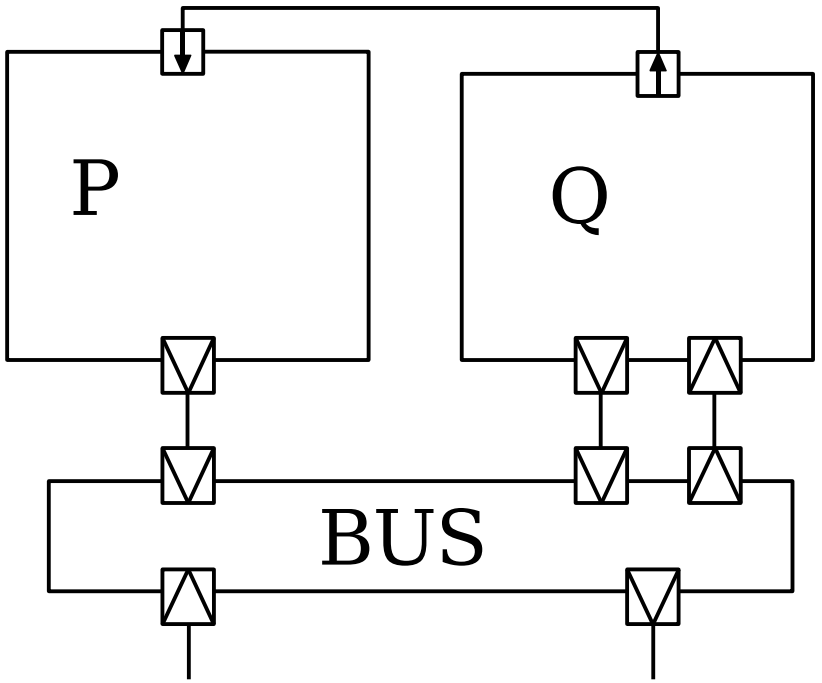
SoC synthesis



RTL

most precise

# SystemC: C++ Library




```
...
unsigned x;
sc_event e;
SC_HAS_PROCESS(top);
top(sc_module_name name):
        sc_module(name) {
  SC_THREAD(P);
  SC_THREAD(Q);
}
void top::P() {
  wait(e);
  ...
```

Construction of the architecture first, then non-preemptive scheduling, simulated time.

# Examples

With fixed delays:

```
void top::P() {
   wait(e);
   wait(20);
   if (x) cout << "Ok\n";
   else cout << "Ko\n";}
```

‖

```
void top::Q() {
   e.notify();
   x = 0;
   wait(20);
   x = 1;}
```

# Examples

Untimed:

```
void top::P() {
     wait(e);
     wait(20);
     yield();
     if (x) cout << "Ok\n";
     else cout << "Ko\n";}
```

‖

```
void top::Q() {
     e.notify();
     x = 0;
     wait(20);
     yield();
     x = 1;}
```

# Examples

With loose delays:

```
void top::P() {
  lwait(3,d1); //t1
  wait(e);
  wait(20); yield();
  lwait(40,d2); //t2
  if (x) cout << "Ok\n";
  else cout << "Ko\n";}
```

||

```
void top::Q() {
  lwait(6,d3); //t3
  e.notify();
  x = 0;
  wait(20); yield();
  lwait(24,d4); //t4
  x = 1;}
```

# The Coverage Problem

- Even if data is fixed
  - The SystemC LRM allows many schedulings
  - Delays may be not fixed (designer choice)
- For the validation of SoC models:
  - 1 execution ⇒ very poor coverage
  - Random schedulings and timings => uncertain coverage, lots of useless executions
  - Test with all possible values => unrealistic
  - Our solution : **test only executions that may lead to different final states**
    - use of Dynamic POR + Linear Programming

# Outline

- Introduction
- Dynamic Partial Order Reduction for Scheduling Generation [1,2]
- DPOR + Linear Programming for schedulings and timings
- Implementation and results
- Conclusion

*[1] P.Godefroid, C.Flanagan, POPL'05*
*[2] C.H., F.M., L.M-C., M.Moy, FMCAD'06*

# Example of Scheduling Dependencies

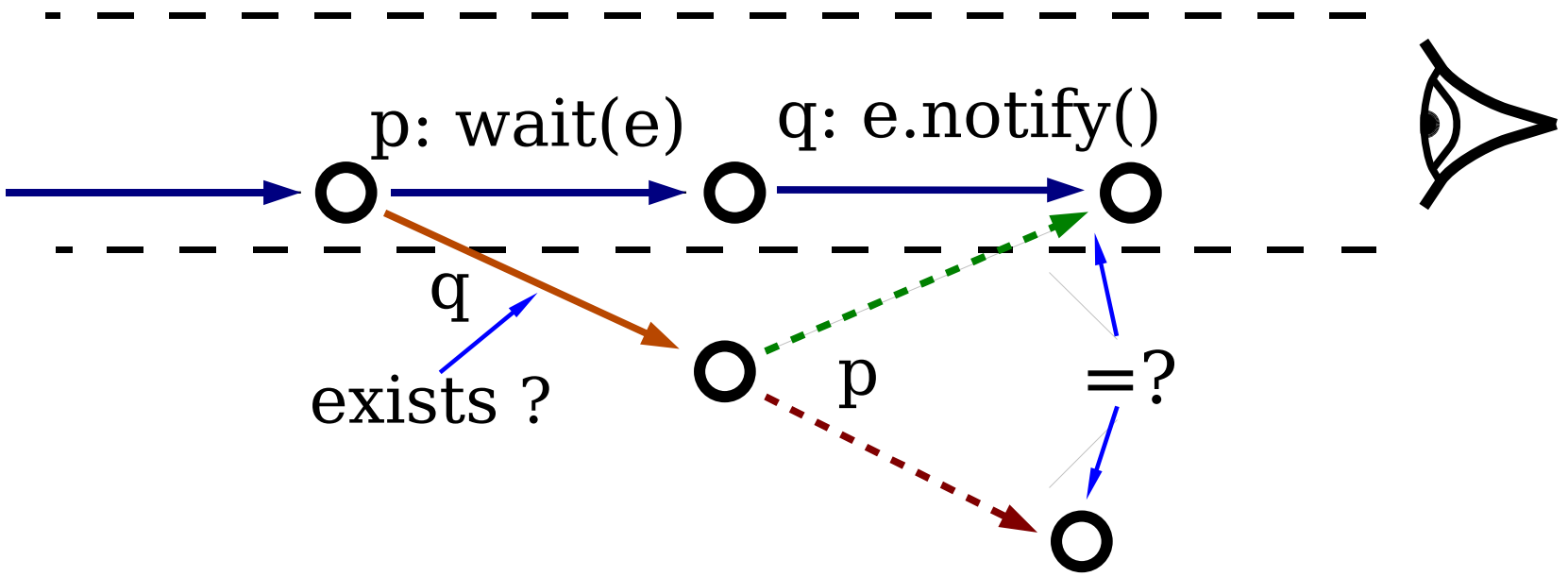
```
void top::P() {
   wait(e);
   wait(20);
   if (x) cout << "Ok\n";
   else cout << "Ko\n";}
```

‖

```
void top::Q() {
   e.notify();
   x = 0;
   wait(20);
   x = 1;}
```

- 3 possible executions:
  - P;Q;P;[TE];Q;P: **Ok** (default OSCI scheduler choice, if `P` declared before `Q`)
  - P;Q;P;[TE];P;Q: **Ko**
  - Q;P;[TE];Q: **“dead-lock”**

# Communications Observation




Goal:
Guess whether transitions are dependent by observing their behavior

# Dynamic Dependency Graph

**Execution Trace:**

$p_1$: wait(e)

$q_1$: notify(e), modify(x)

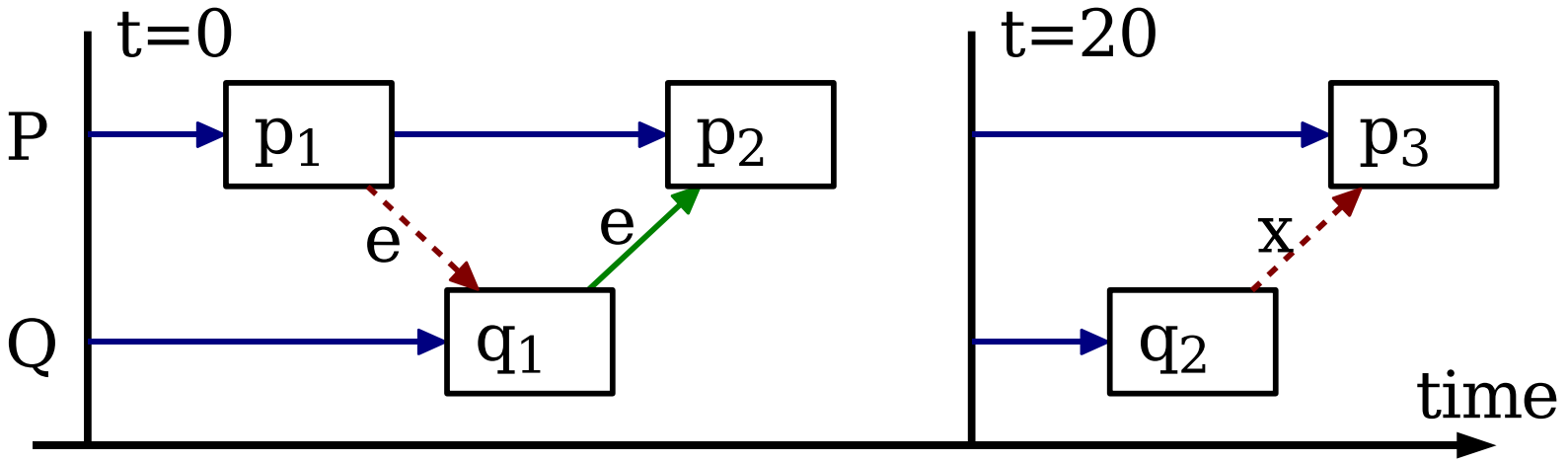
$p_2$: *enabled by $q_1$*

TE: 0 --+20-------->20

$q_2$: modify(x)

$p_3$: read(x)

**Dynamic Dependency Graph:**




**Red arrows imply new executions!**

# Outline

- Introduction
- Dynamic Partial Order Reduction for Scheduling Generation
- DPOR + Linear Programming for schedulings and timings
- Implementation and results
- Conclusion

# Example of Timing Dependencies

```
void top::P() {
   lwait(3,2); //t1
   wait(e);
   lwait(40,10); //t2
   if (x) cout << "Ok\n";
   else cout << "Ko\n";}
```

‖

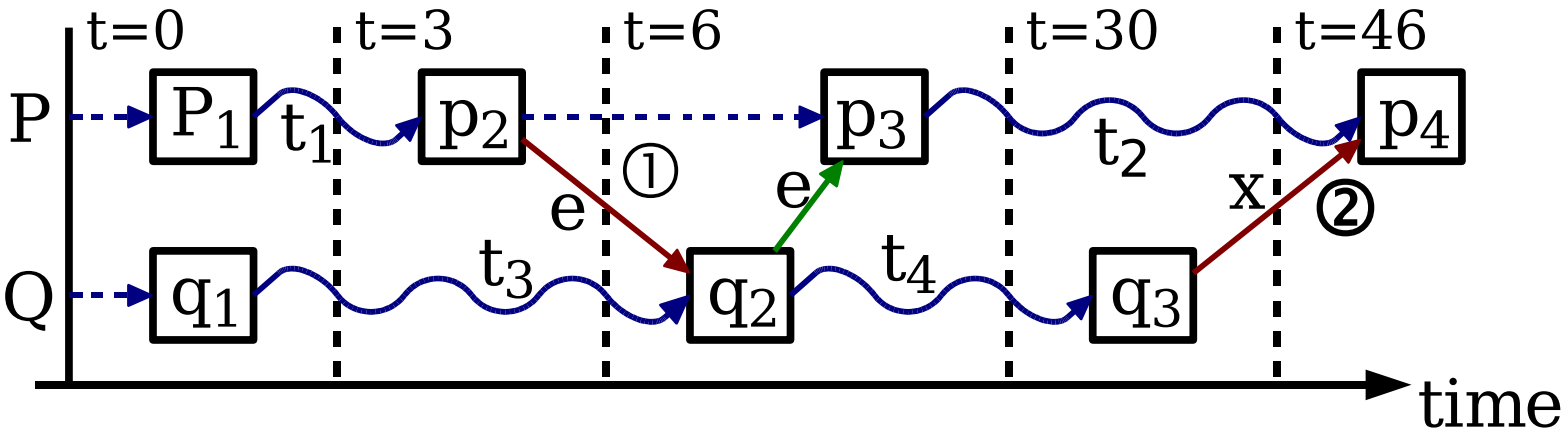
```
void top::Q() {
   lwait(6,2); //t3
   e.notify();
   x = 0;
   lwait(24,6); //t4
   x = 1;}
```

- 3 possible executions again:
  - With $t_1 \rightarrow 3$, $t_2 \rightarrow 40$, $t_3 \rightarrow 6$, $t_4 \rightarrow 24$: **Ok**
  - With $t_1 \rightarrow \mathbf{5}$, $t_2 \rightarrow 40$, $t_3 \rightarrow \mathbf{4}$, $t_4 \rightarrow 24$: **dead-lock**
  - With $t_1 \rightarrow 3$, $t_2 \rightarrow \mathbf{30}$, $t_3 \rightarrow 6$, $t_4 \rightarrow \mathbf{30}$: **Ko** possible

# Example of Timing Generation

- Dynamic Dependency Graph:

DPOR




timing: $t_1 \rightarrow 3$, $t_2 \rightarrow 40$, $t_3 \rightarrow 6$, $t_4 \rightarrow 24$

Two Linear Programs to solve:

LP

① $q_2$ before $p_2$: $t_3 \leq t_1$, $t_1 \in [1,5]$, $t_3 \in [4,8]$

② $p_2$ before $q_2$: $t_3 \geq t_1$, $t_1 \in [1,5]$, $t_3 \in [4,8]$

$p_4$ before $q_3$: $t_2 \leq t_4$, $t_2 \in [30,50]$, $t_4 \in [18,30]$

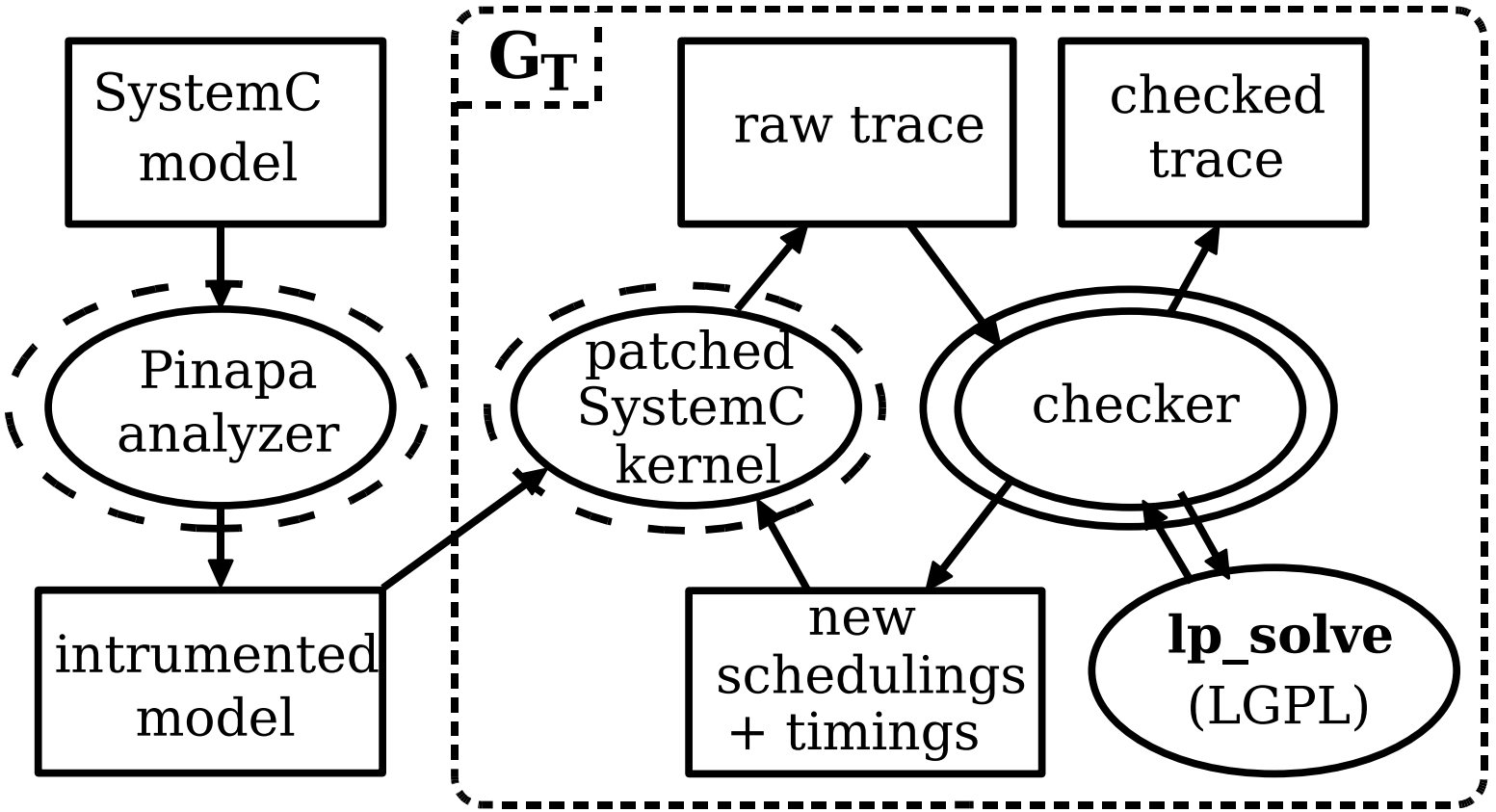
# Constraints Generation

- Symbolic date of a transition $p_i$
  - If enabled by a transition $q_j$ (notification):
    - $sdate(p_i) = sdate(q_j)$
  - If follows a **`lwait(T)`** instruction
    - $sdate(p_i) = sdate(p_{i-1}) + X$
      with X: new variable
- For each scheduling constraint “$p_i$ before $q_j$”:
  - Timing constraint: $sdate(p_i) \leq sdate(q_j)$
- Range of time variables: $T \pm \Delta$

# Constraints Solving

- We get a linear program with:
  - 1 variable per **`lwait`** call
  - 1 constraint per pair of dependent permutable transitions (+ variable ranges)
  - Lots of null coefficients
- We need to exhibit a solution, not only emptyness
- **Solvable without abstraction** using the Simplex Algorithm (first phase only)

# The Tool Chain



# Industrial Case Study : LCMPEG

- 5 components, runs of 150 transitions,
  with long sections of sequential code (~50klines)
- Fixed Delays:
  - 128 schedulings, 1 min 08 sec
- Loose Delays +/- 20%:
  - 3584 executions, 35 min 11 sec
- Untimed version:
  - About 2^32 executions needed, failed.

# Conclusion

- **Working** on medium-sized SoC models
  - With timing variation of about **20%**
- Possible improvements
  - On constraints generation and solving
    - Take 1/3 of the total time on the case study
  - Reduction of the explored space
- Further works
  - Treat more SystemC structures
  - Application to other contexts