Modélisation Transactionnelle des Systèmes sur Puce avec SystemC
Ensimag 3A — filière SLE
Grenoble-INP

Notions Avancé en SystemC/TLM

Julie Dumas
(transparents originaux : Jérôme Cornet, puis Matthieu Moy)

julie.dumas@univ-grenoble-alpes.fr

2017-2018

Planning approximatif des séances

1. Introduction : les systèmes sur puce
2. Introduction : modélisation au niveau transactionnel (TLM)
3. Introduction au C++
4. Présentation de SystemC, éléments de base
5. Communications haut-niveau en SystemC
6. Modélisation TLM en SystemC
7. TP1 : Première plateforme SystemC/TLM
8. Utilisations des plateformes TLM
9. TP2 (1/2) : Utilisation de modules existants (affichage)
10. TP2 (2/2) : Utilisation de modules existants (affichage)
11. Notions Avancées en SystemC/TLM
12. TP3 (1/3) : Intégration du logiciel embarqué
13. TP3 (2/3) : Intégration du logiciel embarqué
14. TP3 (3/3) : Intégration du logiciel embarqué
15. 05/01: Intervenant extérieur : Laurent Maillet-Contoz (STMicroelectronics)
16. Perspectives et conclusion

Grenoble INP ensimag

Sommaire

Grenoble INP ensimag

Sommaire

Grenoble INP ensimag

What is a bug?

1. Launch a SystemC/TLM simulation
2. It produces incorrect result

Question

Good news or bad news?

A Few Kinds of Model/Simulator's Bugs

- Hardware bug:
  1. Simulator design bug, corresponding to a real hardware bug
  2. Simulator programming error, irrelevant in real hardware
- Software bug: software doesn't run properly on simulator
  1. Because software is buggy?
  2. Because simulator is not faithful?

Grenoble INP ensimag

A Few Kinds of Model/Simulator's Bugs

- Hardware bug:
  1. Simulator design bug, corresponding to a real hardware bug 
  2. Simulator programming error, irrelevant in real hardware 
- Software bug: software doesn't run properly on simulator
  1. Because software is buggy? 
  2. Because simulator is not faithful? 

del/Simulator's Bugs

Model only.
Will be thrown away
in final product

- Hardware bug:
  1. Simulator design bug, corresponding to a real hardware bug
  2. Simulator programming error, irrelevant in real hardware
- Software bug: software doesn't run properly on simulator
  1. Because software is buggy?
  2. Because simulator is not faithful?

Actual Software.
Will be embedded
in final product

What Can we Expect from the Model?

Software runs on TLM $\Rightarrow$ Software runs on real chip

What Can we Expect from the Model?

Software runs on TLM $\Rightarrow$ Software runs on real chip

$\Leftrightarrow$

Software doesn't run on real chip

$\Rightarrow$ Software doesn't run on TLM

Another Kind of “Bug”

- Up to now:
  - Hardware Model doesn’t work
  - Software doesn’t run on hardware model
- What about:
  - Software does run on hardware model, but not on real chip

Another Kind of “Bug”

- Up to now:
  - Hardware Model doesn’t work
  - Software doesn’t run on hardware model
- What about:
  - Software does run on hardware model, but not on real chip

Hiding bugs $\neq$ Fixing bugs

Sets of Behaviors, Faithfulness

- An ideal TLM model ...
  - Should exhibit all behaviors of the real system
  - May exhibit more behaviors than the real system
  - Should not exhibit “too many” unrealistic behaviors

(Counter) Examples

TLM should exhibit all behaviors of the real system (1/2)

Software CPU1

```
compute_img(&buf);
img_computed = 1;
```

Software CPU2

```
while (img_computed != 1)
  continue;
read_img(&buf);
```

Question

Are all interleavings correct?

(Counter) Examples

TLM should exhibit all behaviors of the real system (2/2)

(Incorrect) Software CPU1

```
img_computed = 1;
compute_img(&buf);
```

Software CPU2

```
while (img_computed != 1)
  continue;
read_img(&buf);
```

Question

Are all interleavings correct?

(Counter) Examples

TLM should exhibit all behaviors of the real system (2/2)

(Incorrect) Software CPU1

```
img_computed = 1;
compute_img(&buf);
```

Software CPU2

```
while (img_computed != 1)
  continue;
read_img(&buf);
```

Question

Are all interleavings correct?

Question

Will we see the bug in a simulation?

(Counter) Examples

TLM may exhibit more behaviors than the real system

Software CPU1

```
compute_img(&buf);
img_computed = 1;
```

Software CPU2

```
count=0;
while (img_computed != 1)
  count++;
assert(count == 3);
read_img(&buf);
```

(Counter) Examples

TLM should not exhibit “too many” unrealistic behaviors

Software CPU1

```
dest = stg_fast();
```

Software CPU2

```
stg_very_slow();
do_stg_with(dest);
```

- No explicit synchronization between `dest = ...` and access to `dest` ...
- ... but do we want to see this bug?

Set of Behaviors and Non-Determinism

- TLM models should exhibit several behaviors
- Several possibilities $\Rightarrow$ non-determinism
- Implementing non-determinism:
  - Formal verification approach: exhaustive exploration
  - Simulation approach: random

An Example of Non-Determinism: Loose Timing

```
// generates a pseudo-random float
// between 0.0 and 0.999...
float randfloat()
{
    return rand()/(float(RAND_MAX)+1);
}

// loose timing
void pv_wait (x) {
    wait(x*(randfloat()+0.5));
}
```

Question

What does it do?

Sommaire

Sommaire de cette section

Transaction bloc

Avant

```
for (ensitlm::addr_t ad = ad0;
     ad < ad0+size; ad++) {
    write(socket, ad,
          data[ad]);
}
```

Après

```
// 1 echange atomique
block_write(socket, ad, data, size);
```

- Grosse granularité
- Beaucoup plus rapide en simulation

Transaction bloc

Avant

```
for (ensitlm::addr_t ad = ad0;
     ad < ad0+size; ad++) {
    write(socket, ad,
          data[ad]);
}
```

Après

```
// 1 echange atomique
block_write(socket, ad, data, size);
```

- Grosse granularité
- Beaucoup plus rapide en simulation

Question

Perd-t-on en fidélité ?

Sommaire de cette section

Timing approximé et découplage temporel

- Constat : les changements de contexte des *threads* sont lents [1]
- Conséquence 1 : `wait` coûte cher !
- Conséquence 2 : on évite de mettre des `wait`.

1. Malgré l'utilisation des quick threads ou des pthreads

Timing approximé et découplage temporel

- Problème :
  - En PVT, granularité de temps fine.
  - $\Rightarrow$ 1 `wait` pour chaque avancement du temps.
  - $\Rightarrow$ simulation lente.

Timing approximé et découplage temporel

- Problème :
  - En PVT, granularité de temps fine.
  - $\Rightarrow$ 1 `wait` pour chaque avancement du temps.
  - $\Rightarrow$ simulation lente.
- Solution proposée en TLM 2:
  - Chaque processus peut être « en avance » sur le temps global.
  - (L'avance correspond à l'argument `sc_time` des méthodes `transport` de TLM-2, ignoré dans ENSITLM)
  - Quand faire les `wait()` (i.e. laisser le reste de la plate-forme rattraper notre avance) ?
    - ⋆ Quantum Keeping: Si on est plus « en avance » que le quantum (constante de temps choisie par l'utilisateur)
    - ⋆ Synchronisation explicite: Avant (ou après ?) les points de synchronisation

Timing approximé et découplage temporel

Exemple

```
sc_time t = SC_ZERO_TIME; // local advance over SystemC time
local_computation(); // Instantaneous wrt SystemC
t += sc_time(12, SC_NS); // don't wait() now
other_computation();
t += sc_time(3, SC_NS);
socket.write(addr, data, t); // may update t
wait(t); t = SC_ZERO_TIME; // Catch-up with SystemC time
send_interrupt();
```

Timing approximé et découplage temporel

Exemple

```
sc_time t = SC_ZERO_TIME; // local advance over SystemC time
local_computation(); // Instantaneous wrt SystemC
t += sc_time(12, SC_NS); // don't wait() now
other_computation();
t += sc_time(3, SC_NS);
socket.write(addr, data, t); // may update t
wait(t); t = SC_ZERO_TIME; // Catch-up with SystemC time
send_interrupt();
```

Question

Quels sont les problèmes ?

Sommaire de cette section

Parallélisation de SystemC

- Paradoxe:
  - Les systèmes sur puces sont parallèles
  - SystemC a une notion de processus
  - SystemC n'exploite qu'un processeur !

Parallélisation de SystemC

- Paradoxe:
  - Les systèmes sur puces sont parallèles
  - SystemC a une notion de processus
  - SystemC n'exploite qu'un processeur !
- Solution (très) naive:
  - 1 SC_THREAD $\rightarrow$ 1 pthread
  - On lance tout en parallèle.
  - $\Rightarrow$ beaucoup de pthreads, ne passe pas à l'échelle.

Parallélisation de SystemC

- Paradoxe:
  - Les systèmes sur puces sont parallèles
  - SystemC a une notion de processus
  - SystemC n'exploite qu'un processeur !
- Solution (très) naive:
  - 1 SC_THREAD $\rightarrow$ 1 pthread
  - On lance tout en parallèle.
  - $\Rightarrow$ beaucoup de pthreads, ne passe pas à l'échelle.
- Solution moins naive:
  - N processeurs $\rightarrow \approx N$ pthreads.
  - Gestion du mapping « processus SystemC » $\leftrightarrow$ pthread dans le kernel SystemC.

Parallélisation de SystemC

- Paradoxe:
  - Les systèmes sur puces sont parallèles
  - SystemC a une notion de processus
  - SystemC n'exploite qu'un processeur !
- Solution (très) naive:
  - 1 SC_THREAD $\rightarrow$ 1 pthread
  - On lance tout en parallèle.
  - $\Rightarrow$ beaucoup de pthreads, ne passe pas à l'échelle.
- Solution moins naive:
  - N processeurs $\rightarrow \approx N$ pthreads.
  - Gestion du mapping « processus SystemC » $\leftrightarrow$ pthread dans le kernel SystemC.

Question

Où est le problème ?

Parallélisation de SystemC

- Si on veut faire les choses proprement:
  - Analyse statique des dépendances de données
  - Prise en compte à l'exécution
  - (e.g. DRT de Yussef Bouzouzou, 2007—2 ans de travail)
  - $\Rightarrow$ on sort du principe « SystemC, c'est facile à compiler, g++ le fait très bien ».
- Problème restant:
  - Parallélisation « à l'intérieur du δ-cycle », mais exécuter en parallèle des processus censés s'exécuter à différents instants de simu = difficile.
- Solutions envisageables:
  - Profiter du découplage temporel pour paralléliser
  - Tâches avec durée (cf. jTLM, et maintenant sc_during)

Parallélisation de SystemC : conclusion

- C'est dur.
  PDES (*Parallel Discrete Event Simulation*)
  théorie établie entre 1979 (Chandy-Misra, Lamport) et 1990 (Fujimoto)
  implantations peu efficaces pour nos problèmes
- La plupart des gens qui le font ne se soucient pas de préservation de la sémantique
- Solution en pratique : lancer N simulations sur $< N$ machines !
- Il faut peut-être un autre langage ?

Accélération des simulations SystemC

- Code interne aux composants = C++
  $\Rightarrow$ `g++ -O3` et le tour est joué (ou pas)
- Context-switch = cher :
  - Scheduling
  - Sauvegarde/restauration de tous les registres
  - Changement du pointeur de pile $\Rightarrow$ cache-miss
  - Changement du compteur programme $\Rightarrow$ vidage de pipeline
  - sans compter les flushs de TLB, etc, suivant la manière dont les threads sont implantés
- Transactions = cher:
  - Plusieurs appels de méthodes virtuelles ($\Rightarrow$ non inline-ables)
  - Décodage d'adresse pour router la transaction (en O(nb slave))
  - $\Rightarrow$ Là où la vraie plateforme fait un `load`/`store`, le code TLM exécute du code difficile à optimiser.

Accélération des simulations SystemC
Minimiser le coût du au context-switch

- Utilisation de `SC_METHOD` à la place des `SC_THREAD` (pas toujours possible)
- Minimisation du nombre de `wait` à exécuter
- Scheduling statique (`http://www.cprover.org/scoot/`: plus de travail à la compilation, simulation 2 à 6 fois plus rapide sur des exemples)

Accélération des simulations SystemC
Minimiser le coût du aux transactions

- DMI = Direct Memory Interface: on récupère un pointeur sur la zone mémoire intéressante, et on fait des accès sans passer par le bus.

Accélération des simulations SystemC
Minimiser le coût du aux transactions

- DMI = Direct Memory Interface: on récupère un pointeur sur la zone mémoire intéressante, et on fait des accès sans passer par le bus.

Question

Quel est le problème ?

Accélération des simulations SystemC
Minimiser le coût du aux transactions

- DMI = Direct Memory Interface: on récupère un pointeur sur la zone mémoire intéressante, et on fait des accès sans passer par le bus.

Question

Quel est le problème ?

- Techniques de compilation spécifiques : lancer une passe d'optimisations après l'élaboration (prototype basé sur LLVM développé au LIAMA et à Verimag)

Sommaire

Sommaire de cette section

Question

Quel est le problème ?

Sommaire de cette section

Vérification Formelle de SystemC

- Plusieurs approches :
  - Compilation de SystemC vers des langages sources des outils de preuve
    - ⋆ Front-end SystemC dédié
    - ⋆ Modélisation formelle de toutes les constructions SystemC
  - Exploration exhaustive de l'espace d'états à l'exécution
    - ⋆ stateless: on explore tous les ordonnancements de taille $< N$
    - ⋆ statefull: on ajoute la possibilité de mémoriser et de comparer des états ($\Rightarrow$ construction de l'espace d'états entier)
  - En général, problème de passage à l'échelle (scalability)
    produits d'automates $\Rightarrow$ explosion de l'espace d'états (state explosion)

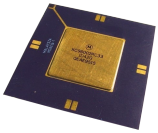
Sommaire

Power and Temperature Estimation

Question

How?

Power estimation in TLM: Power-state Model

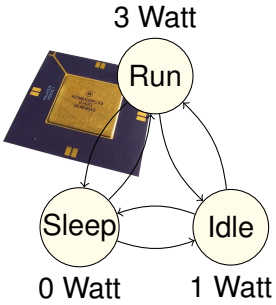


```
// SystemC thread
void compute() {
    while (true) {

        f();
        wait(10, SC_MS);

        wait(irq);
    }
}
```

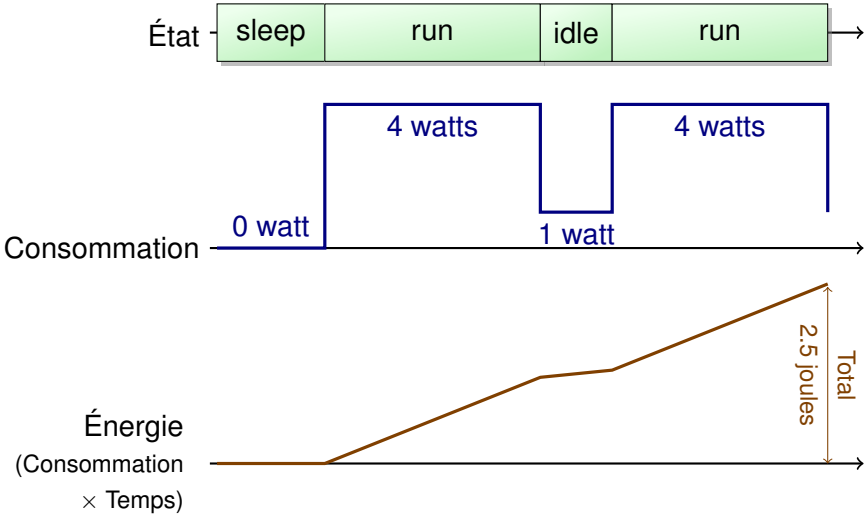
Power estimation in TLM: Power-state Model



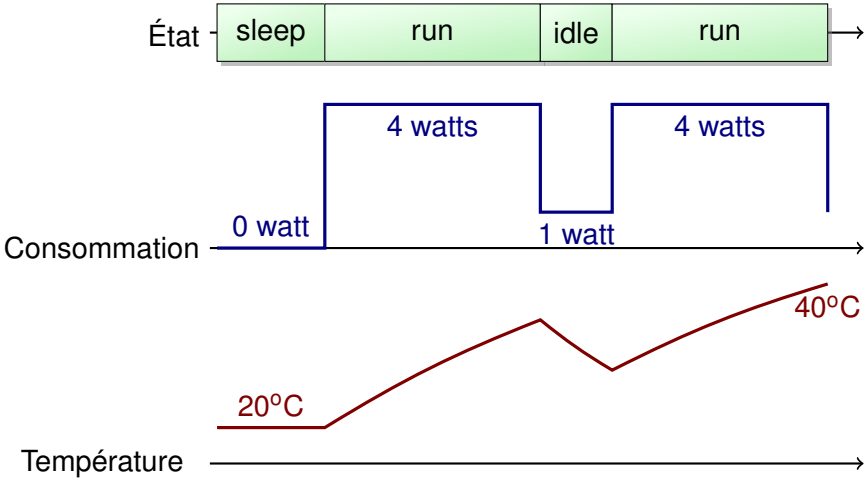

```
// SystemC thread
void compute() {
    while (true) {
        set_state("run");
        f();
        wait(10, SC_MS);
        set_state("idle");
        wait(irq);
    }
}
```

- Consumption depends on:
  - Activity state (switching activity inside component)
  - Electrical state (voltage, frequency)
  - Traffic (stimulation by other components)

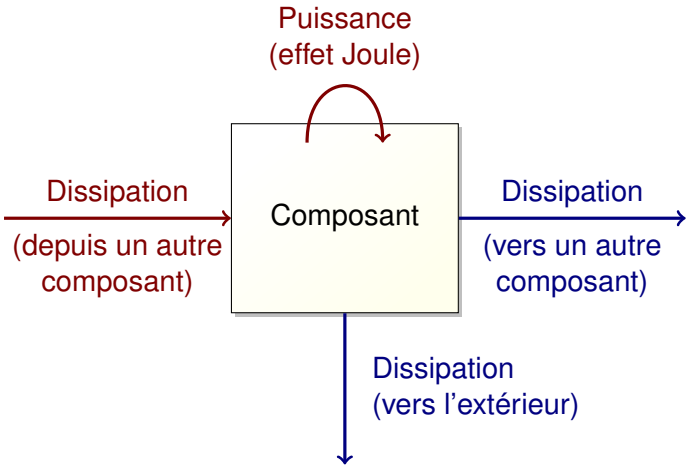
Des états à la consommation



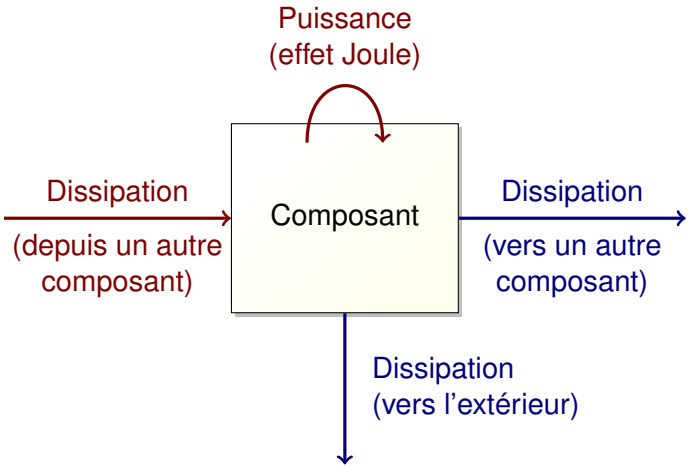
De la consommation à la température



Puissance consommée, température, et dissipation de chaleur

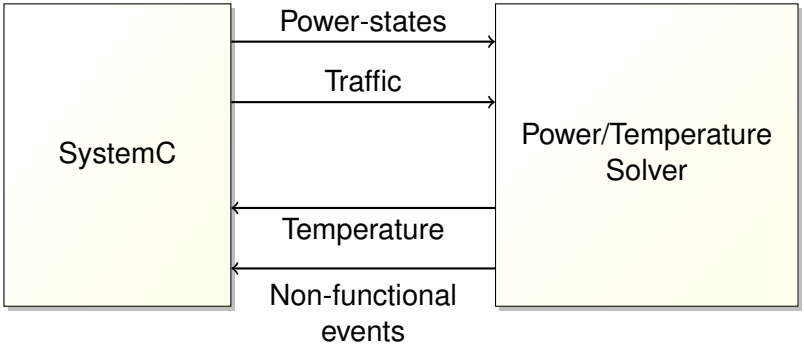


Puissance consommée, température, et dissipation de chaleur



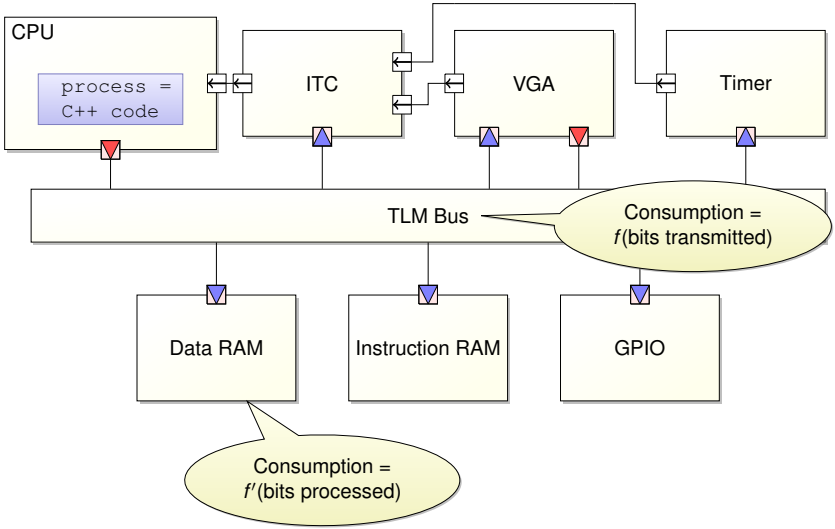
$\rightsquigarrow$ système d'équation différentielles, résolus par des solveurs dédiés

SystemC and Temperature Solver Cosimulation

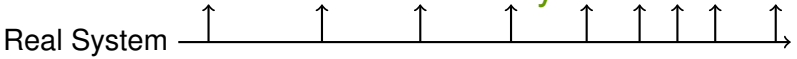


Functionality can depend on non-functional data
(e.g. validate power-management policy)

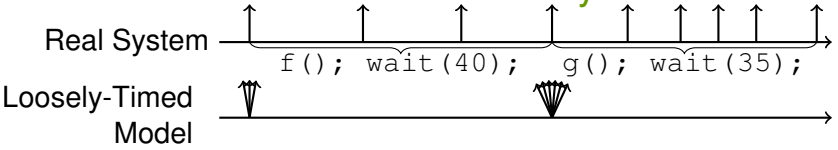
Traffic Models



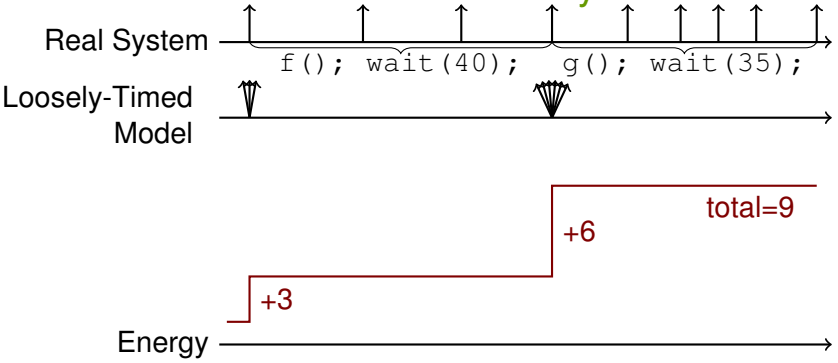
Traffic Model and Loosely Timed Models



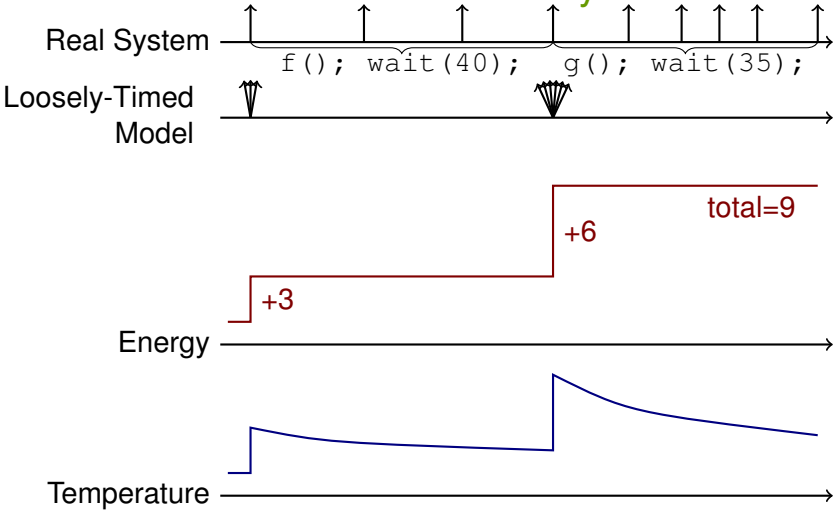
Traffic Model and Loosely Timed Models



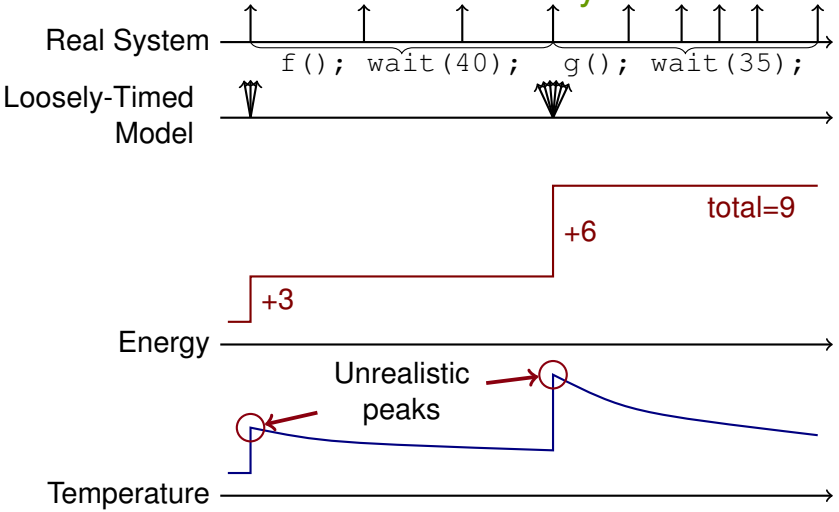
Traffic Model and Loosely Timed Models



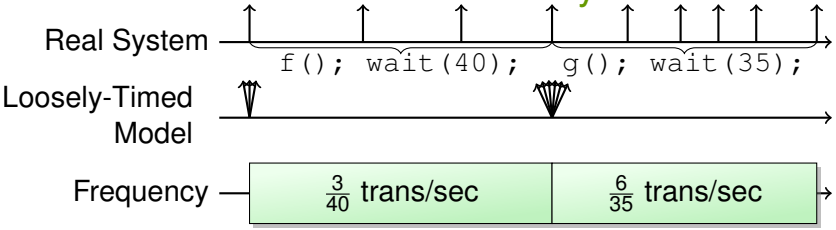
Traffic Model and Loosely Timed Models



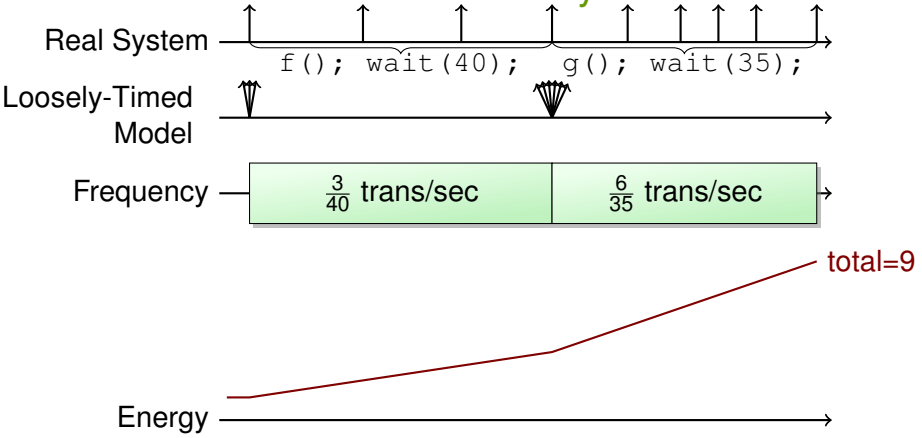
Traffic Model and Loosely Timed Models



Traffic Model and Loosely Timed Models



Traffic Model and Loosely Timed Models



Traffic Model and Loosely Timed Models