MOVES - A tool for Modelling and Verification of Embedded Systems

J. Ellebæk, K. S. Knudsen, A. Brekling, M. R. Hansen and J. Madsen
Embedded Systems Engineering Group
Informatics and Mathematical Modelling, Technical University of Denmark

Abstract

We demonstrate MOVES, a tool which allows designers of embedded systems to explore possible implementations early in the design process.
The demonstration of MOVES will show how designers can explore different designs by changing the mapping of tasks on processing elements, the number and/or speed of processing elements, the size of local memories, and the operating systems (scheduling algorithm).

1. Introduction

One of the major challenges in designing an embedded system is to find a mapping of the application onto the execution platform which effectively fulfills the non-functional requirements of the embedded system such as timing, memory usage, energy consumption, and other cost. A particular challenge is to model and analyse cross-layer dependencies, where the change of a property in one part of the system, e.g. scheduling policy, may impact the performance of another part of the system, e.g. deadline miss on another processor, and hence, the overall system performance.

MOVES is a tool which supports formal analysis of non-functional properties of an embedded system, covering the system layers of an application mapped on an execution platform, consisting of a heterogeneous multiprocessor architecture where each processor may run a real-time operating system, and where all processors are connected through a network. The system-level model of MOVES is based on ARTS[1].

2. System-level model

An application is specified as a set of independent programs which have to be executed on the execution platform. Each program is modelled as a task graph, i.e. a directed acyclic graph where each task is a node and edges indicate causal dependencies (see Figur 1). A task τ_j is currently considered to be periodic and is represented by a period, a deadline and an offset. The execution time and memory footprint of the task will depend on the characteristics of the processing element pe_i to execute it.
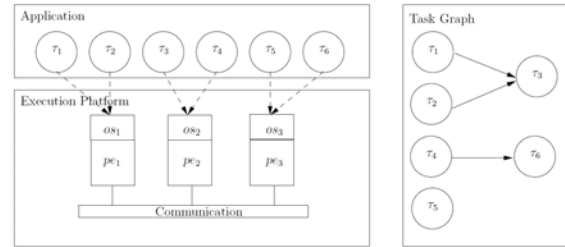
Figur 1 System-levl model for a MPSoC

The execution platform is a heterogeneous multiprocessor System-on-Chip (MPSoC) in which a number of processing elements pe are connected through an on-chip network. A processing element pe_i is characterized by a clock frequency f_i, a local memory m_i with a bounded size, and a real-time operating system os_i which schedules the tasks mapped to the processing element and handles inter-task dependencies.

An application implementation is a mapping of tasks to processing elements of the execution platform. When a task τ_j is mapped to a processing element pe_i it is equipped with worst case $wcet_{ji}$ and best case $bcet_{ji}$ execution times which means that a task may complete its execution anywhere in the interval $[bcet_{ji}; wcet_{ji}]$. The exact values of $bcet_{ji}$ and $wcet_{ji}$ are dependent on the type and frequency f_i of the processing element pe_i. The memory footprint of a task is split into a static part (the program code) and a dynamic part (the data needed during execution and for data transferred to another task).
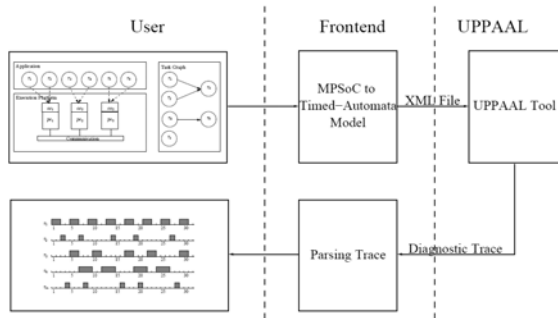
2. MOVES

MOVES is based on timed-automata models of tasks with causal dependencies and the components of the execution platform. The frontend of MOVES will translate a given implementation of an application into an UPPAAL[2] model. The UPPAAL tool environment is used as backend for MOVES, as illustrated in Figur 2.

Currently MOVES supports preemptive periodic tasks, hard and soft deadlines, causal dependencies between tasks on the same as well as on different processing elements, and execution times as discrete values between best and worst case. Systems of arbitrary sizes can be modelled and MOVES offers verification of timing, energy and memory usage. Systems of 250 tasks mapped onto 15 processing

elements have been formally verified for timing and memory properties.

Figur 2 Components of the tool

3. Using MOVES

Designers can explore different designs by changing the mapping of tasks, the number and/or speed of processing elements, the size of local memories, and the operating systems (scheduling algorithm).

The application, execution platform and mapping are given by the user in a simple language, currently embedded within Java, as shown below.

```
public class system {
    public Application apps;
    public Platform p1;
    public system(int M, int N, int granularity) {
    Resource r1 = new Resource();

    //Tasks (bcet, wcet, deadline, offset, period, FP)
    Task t1 = new Task(2, 2, 4, 0, 4, 1);
    Task t2 = new Task(1, 1, 6, 0, 6, 2);
    Task t3 = new Task(2, 2, 6, 0, 6, 3);
    Task t4 = new Task(3, 3, 6, 4, 6, 4);
    Task tm = new Task(1, 1, 6, 0, 6, 5);

    //Processors (frequency, schedule policy)
    Processor p1 = new Processor(1, Processor.RM);
    Processor p2 = new Processor(1, Processor.EDF);
    Processor pm = new Processor(1, Processor.RM);

    //Mapping
    Task[][] tasks = {{t1,t2},{t3,t4},{tm}};

    //Adds the processors to the system
    Processor[] ps = {p1,p2,pm};

    // cost
    Cost memory = new Cost(tasks);
    Cost power = new Cost(tasks);
    Cost[] ca = {memory, power};

    p1 = new Platform(ps);
    apps = new Application(tasks, ca, r1.rs, granularity);

    //Add dependencies to the system.
    apps.addDep(t2,tm);
    apps.addDep(tm,t3);
    apps.useResource(tm,r1);

    //memory usage (task-id, static, idle, redy, running)
    memory.set(t1,1,0,0,3);
    memory.set(t2,1,0,0,5);
    memory.set(t3,2,0,0,6);
    memory.set(t4,1,0,0,9);
    memory.share(t2,t3,5);

    //power usage (task-id, static, idle, redy, running)
    power.set(t1,0,0,0,5);
    power.set(t2,0,0,0,10);
    power.set(t3,0,0,0,10);
    power.set(t4,0,0,0,5);
    }
}
```

This code, together with the property to be verified, is translated into a timed automata model and verified by UPPAAL, as illustrated in Figur 2. If the property is not satisfied, UPPAAL can create the trace of a counter example. MOVES translates this trace into a Gantt chart, in which the designer can see where the system failed. The properties which can currently be verified are:

- Timing: `E<>missedDeadline`
- Memory: `E<>totalCostUsed(Memory)>=23`
- Energy: `E<>totalCostUsed(Energy)>=15`

4. Case study

We have used MOVES to explored implementations of an MP3 decoder. The MP3 decoder has 16 tasks (see Figur 3) performing all the steps for producing an audio stereo frame every 25 ms.
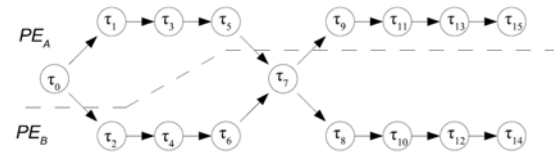
Figur 3 Task graph for the MP3 decoder

The worst case execution times vary from 476 to 266.687 cycles. Analysis has to be applied over the complete interval of 25 ms, which amounts to 25.000 time units using a clock frequency of 25MHz and a granularity of 25 cycles per unit. The granularity is a parameter in MOVES which allows the designer to limit the search space. If the desired property can not be satisfied, the granularity has to be lowered. In the case of the MP3 decoder, MOVES verifies the timing properties on a two processor platform in 30 s.

5. References

[1] S. Mahadevan, M. Storgaard, J. Madsen, K. Virk, *ARTS: A system-level framework for modeling MPSoC components and analysis of their causality,* In proceedings og the 13th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS), 2005, 480-483.

[2] G. Behrmann, A. David, K. G. Larsen, *A tutorial on uppaal*, Lecture Notes in Computer Science 3185 (2004) 200-236.

[3] A. Brekling, *Modelling and Verification of MPSoC,* Master thesis, Informatics and Mathematical Modeling, technical University of Denmark, December 2006.