CLASSY user manual

Xin An and Abdoulaye Gamatié

October 9, 2012

1 Introduction

CLASSY is an abstract clock based system design and validation framework providing:

- rapid prototyping (simulation and analysis) of system design
- design space exploration of application mappings
- on-line simulation and analysis of adaptive system design

The tool is developed by using Java, and to use it, one requires to deploy the Java Runtime Environment, also known as Java Virtual Machine (JVM). Finally, unzipping classy.zip and typing in a terminal: java -jar classy.jar, you will see the welcome window (see Figure 1).

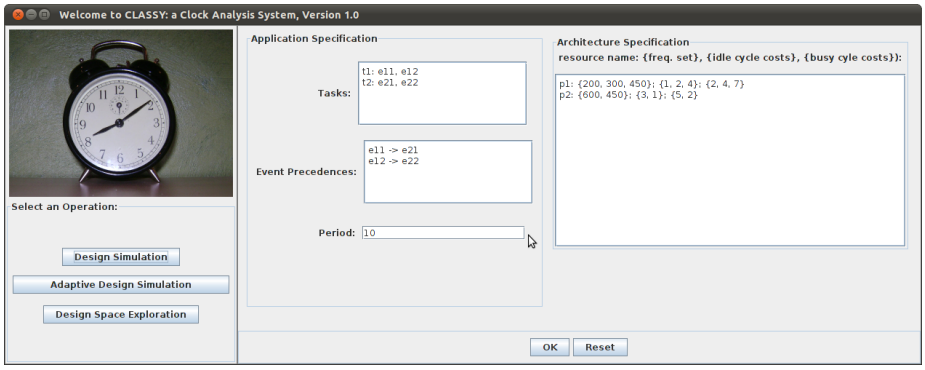

Figure 1: The welcome window of CLASSY

In the following sections, we firstly introduce the welcome window, including its general input in Section 2. The principles of abstract clock modeling of system behavior and executions are briefly described in Section 3. We then illustrate how to use CLASSY to perform the above functionalities respectively.

2 Welcome Window

The welcome window (see Figure 1) consists of two panels. The left panel provides simply the entrances to corresponding functionalities. The user specifies the studied system on the right panel. It consists of two parts: the application and architecture specifications. We consider an application as a set of tasks, and a task as a sequence of events. The user can also define the precedence relations between events of different tasks e.g., data produce and consume events. The application operates periodically. We consider the architecture as a set of processing elements (PE). Each PE is characterized by possible processing frequencies, and corresponding energy consumptions per idle and busy cycles respectively.

3 Clock Modeling of System Behavior and Execution

We briefly introduce our clock modeling of system behavior and executions. This would help users to better understand the tool and its computing results. We refer the users to [1] for more details.

The modeling of application behavior as described in Section 2 directly follows the application specification in the welcome window. Figure 2 illustrates an application behavior example, which consists of tasks $\{t_0, t_1, t_2\}$. Tasks t_0, t_1, t_2 are modeled as sequences of events $\{e_0^0, e_0^1\}, \{e_1^0, e_1^1\}, \{e_2^0, e_2^1\}$ respectively. The *arrows* in the figure are used to represent the precedence relations between events. For example, the arrow from event e_0^0 to event e_2^0 represents $e_0^0 \prec e_2^0$. The absence of arrow connection between two events means no precedence constraint between them, e.g., events e_0^1 and e_1^0.

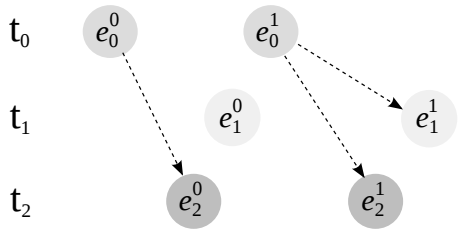

Figure 2: An application behavior b_T

We consider the execution platform consisting of a set P of PEs operating synchronously according to a reference clock and communicating via a shared memory. We model platform behaviors through their clock activations according to given frequency values f_i of processing elements $p_i \in P, 1 \leq i \leq |P|$. The frequency value of the reference clock $\mathcal{K}$ of the platform is calculated as $LCM(f_1, ..., f_{|P|})$, where LCM denotes the Least Common Multiple. More con-

cretely, the clock activation instants of the processing elements are modeled within a trace by considering the inverse of frequency values $1/f_i$, i.e., their period values. They are also referred to as processing element *clock cycles* in our approach. Figure 3 illustrates the behavior of a platform composed of three processors p_0, p_1 and p_2 with frequencies $f_0 = 60MHz, f_1 = 40MHz$ and $f_2 = 30MHz$.

	0	1	2	3	4	5	6	7	8	...
$\mathcal{K}$	•	•	•	•	•	•	•	•	•	...
p_0	•		•		•		•		•	...
p_1	•			•			•			...
p_2	•				•				•	...

Figure 3: Clock trace of PEs

We model the executions of tasks on PEs by means of a *ternary abstract clock* encoding. Such an abstract clock is a three-valued string over $\{-1, 0, 1\}$. The values 1 and 0 respectively represent the *active* and *idle* instants of a processing element executing some tasks w.r.t. the reference clock. The meaning of the value -1 is contextual: a sequence of -1 means active at these instants if it is preceded by 1, otherwise it denotes idle. Figure 4 shows three ternary clocks, representing the executions of the tasks given previously in Figure 2 on the PEs of Figure 3. In this example, tasks t_0, t_1 are executed on p_0, while t_2 on p_1, and all events require one PE cycle to process. Take the ternary clock denoted by $clk(t_0/p_0)$ (representing execution of task t_0 on PE p_0) as an example, the execution of event e_0^0 (resp. e_0^1) starts from the very first (resp. 4th) instant of the reference clock, and takes one clock cycle of p_0.

	0	1	2	3	4	5	6	7	8	9
p_0	•		•		•		•		•	
$clk(t_0/p_0)$	1	-1	0	-1	1	-1				
$clk(t_1/p_0)$	0	-1	1	-1	0	-1	1	-1		
p_1	•			•			•			•
$clk(t_2/p_1)$	1	-1	-1	0	-1	-1	1	-1	-1	

Figure 4: Task schedules in terms of ternary clocks

The execution clocks represent how the tasks execute on PEs, and can be used to generate vcd files to feed the graphical tool GTKWave to visulize the results. The execution time of an application is the maximal duration of execution clocks. Given the scheduling clocks and energy costs per busy and idle cycles for PEs, energy consumption can be computed easily. However, computation of execution clocks also takes much more time. It is not recommended if the user only wants the performance results.

4 Rapid Prototyping of System Design

This functionality allows the user to simulate and analyze design (or mapping) choices. Provided the general system specification from Section 2, the user requires to, accordingly, define his/her design choice (see Figure 5):

- mapping: choose a processing element for each task;
- processing frequencies: choose a frequency for each allocated processing element;
- deadlines (optional): define the deadlines of tasks;
- profiling data: associate computation and communication time costs with each event executing on its allocated resource.

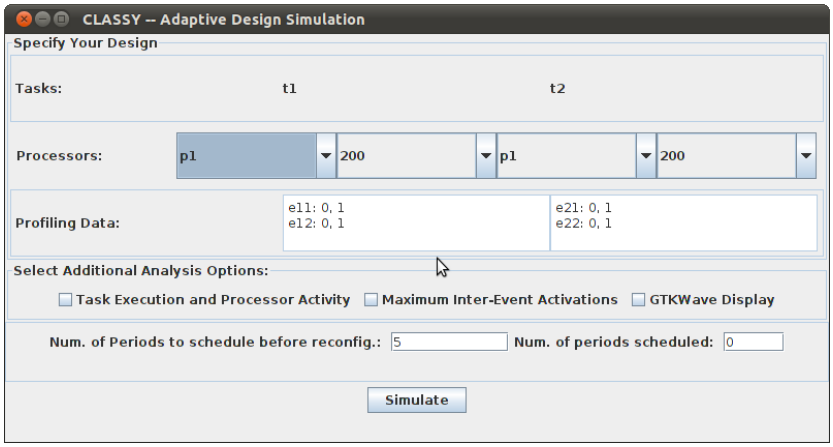

Figure 5: The design analysis window

By pressing button "Analysis", the user gets the default analysis result as shown in Figure 6.

Further analysis options are available as seen in Figure 5:

- Task Execution and Processor Activity: generate the scheduling clocks illustrating task execution and processor activity behaviors;
- Maximum Inter-Event Activations: regarding each defined precedence relation $e_i -> e_j$, compute the maximal number of precedent event activations (i.e., e_i) between e_i and e_j. This indicates how many e_i, at most, have been executed before its successor e_j becomes active.
- GTKWave Display: generate the vcd file to feed GTKWave for display, allowing the user to have a better vision. Figure 7 gives an example.

The user can change its design choice and then re-analysis by pressing button "Reset" and "Analysis".

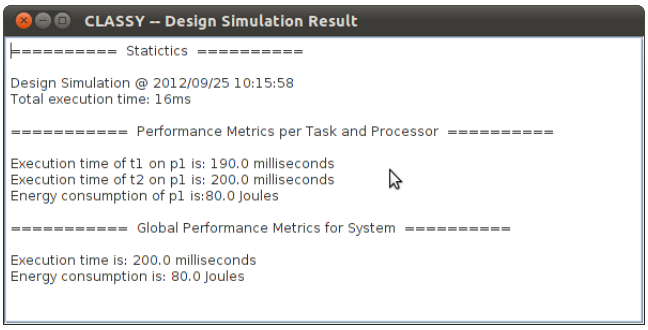

Figure 6: The default simulation output window

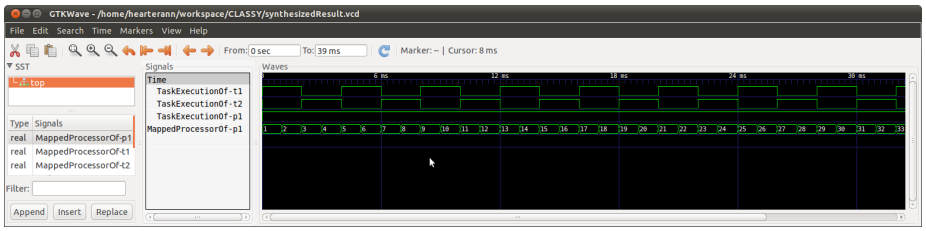

Figure 7: The GTKWave display of Design in Figure 5

5 Adaptive Design Simulation

This functionality provides a reactive simulation environment to enable the simulation and analysis of adaptive system designs. CLASSY deals with two types of adaptive events: changing frequencies of PEs and migrating tasks among PEs. And it takes into account a adaption penalty for each adaptive event.

As shown in Figure 8, the user needs to firstly define the initial mapping choice and corresponding profiling data. And then, before starting the simulation process, he/she also needs to define the next predicted reconfiguration point, such that allowing the tool to react in time along the simulation process. In CLASSY, a reconfiguration point is defined as end of executions of a certain number of period.

For example, we keep the default mapping and profiling data in Figure 8, and would like to reconfigure after the execution of 5 periods, by specifying 5 in the text field after "Num. of Periods to schedule before reconfig.:". By pressing button "simulate", the tool outputs the (temporary) analysis result (see Figure 9) for current configuration, and allows the user to adapt the system execution by changing frequency or migrating tasks. As an example, we migrate task $t2$ to PE $p2$ with frequency 600, and once pressing button "simulate", the tool asks for the operation penalty (see Figure 10) before performing the scheduling based on the newly defined configuration. This procedure continues until the complete simulation finishes and results reported accordingly.

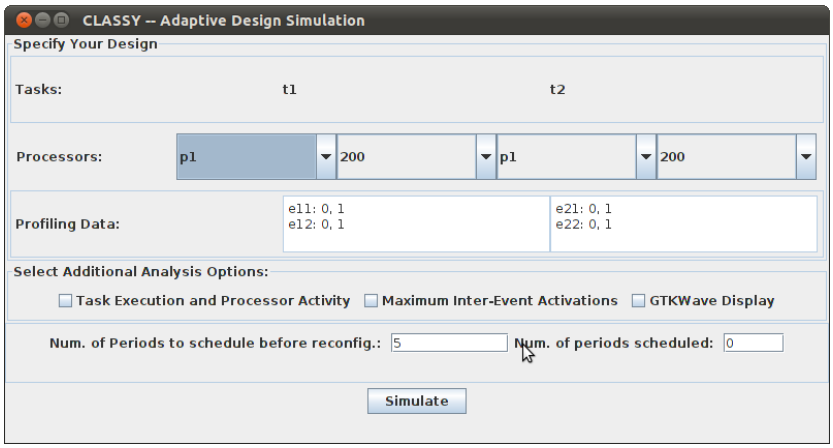

Figure 8: The adaptive design simulation

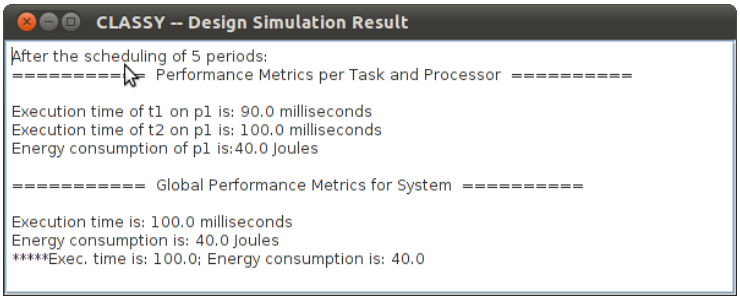

Figure 9: The adaptive design temporary result

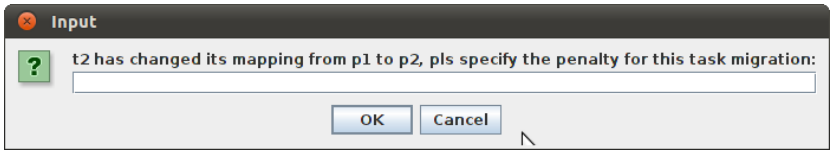

Figure 10: The penalty definition window

6 Design Space Exploration (DSE)

This functionality assists the user to find out a set of Pareto-optimal design choices w.r.t. execution time and energy consumption. It employs two exploration methods: an exhaustive and an Evolutionary Algorithm (EA)-based. The user only needs to define for each task all the profiling data, i.e., the communication and computation cycle costs on possible PEs, before choosing from the provided DSE operations by clicking buttons "Exhaustive DSE" or "Heuristic DSE" (see Figure 11).

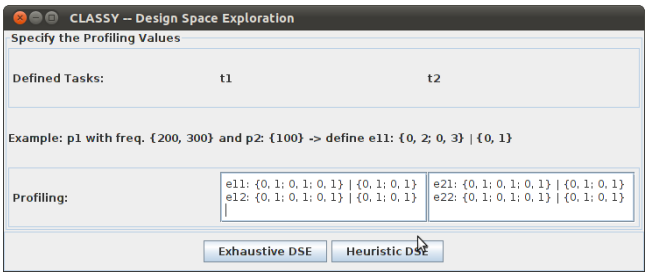

Figure 11: The DSE window

In the following, we introduce these two DSE operations:

- exhaustive DSE: this functionality is achieved by exploring the design space, i.e., mappings and frequencies, exhaustively, and finds out the Pareto-optimal choices.
- heuristic DSE: this functionality assists the user to find a set of Pareto-(sub)optimal solutions efficiently when the system design space is too big, or even huge such that exhaustive exploration takes too much time and resources that the user doest not want to afford. CLASSY employ the evolutionary algorithm NSGA-II [2] to achieve this. This is done by combing CLASSY with the jMetal framework [3], which implements a number of multi-objective algorithms including NSGA-II. The user can define parameters, i.e., population size and iteration times (see Figure 12), to impose the resulting solution size and iteration times of computations.

7 Conclusion and Perspectives

This document describes the functionalities provided by the CLASSY tool, and how a user can use it. Among the possible improvements, we consider the following ones in the near future:

- take into account infeasible solutions: some tasks probably can only run on a subset of PEs, thus these constrains should be considered when performing random adaptation and DSE;

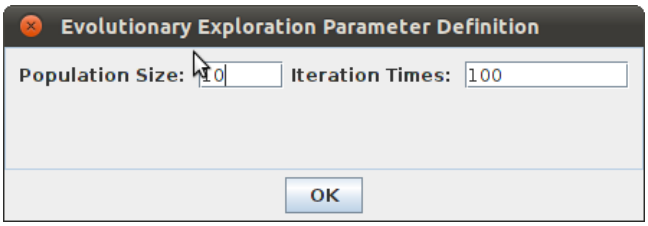

Figure 12: The evolutionary algorithm parameter definition

- provide more scheduling policies/algorithms;
- deal with task adaptations, i.e., a task changing its implementation in response to environment, this is reflected in our simulation framework by changing the profiling data (if composed events defined properly);
- make the tool more user friendly.