# A HIGH-LEVEL BASED FRAMEWORK APPROACH FOR DESIGNING RECONFIGURABLE SYSTEMS

*Remy Eskinazi*
UPE - Escola Politécnica, CEFET-PE
remy.eskinazi@gmail.com

*Manoel Eusebio, Halmos Fernando, Abel Guilhermino, Paulo Maciel, Stelita Silva, Jordana Seixas, Pablo de Santana, Paulo Sergio B. Nascimento*
UFPE – Cin
{mel, hfn, agsf, prmm, sms, jls, psb, psbn}@cin.ufpe.br

## ABSTRACT

This paper presents a high-level framework for designing real-time reconfigurable systems. The framework has two major stages. The first stage is real time task scheduling generation in order to deal with the task timing constraints. The second stage is a SystemC description that is used to model the scheduling and a FPGA reconfigurable system. Experimental results are presented in order to validate the methodology.

## 1. INTRODUCTION

During the last years, the development of embedded systems has more and more considered run-time reconfiguration. This fact occurs mainly because of the enormous advances in the implementation of digital electronic systems, especially the ones presenting some kind of embedded computation [1], [2], [3]. Considering partial and dynamic reconfiguration, this work presents a framework that enables the design of FPGA based reconfigurable systems. This paper is organized as follows: Section 2 presents an overview about recent works on run-time reconfiguration. Section 3 brings the framework description, with detailed explanation on its internal representation, scheduling generation, tasks mapping and execution of the configuration. Section 4 describes the high-level reconfigurable modeling in SystemC. Section 5 describes in details an example and results. Section 6 concludes this paper and presents some direction for future works.

## 2. RELATED WORK

Dynamically Reconfigurable FPGAs have occupied a large space in designing modern embedded systems. This fact comes from the increment of reconfigurable blocks available and powerful soft-core embedded processors. However, methods for efficient implementation of run-time systems based on these components are still a large field of research. Problems such as partitioning, task allocation and scheduling have been arduously studied due to the difficulty in run-time system implementation and the necessity for increasing system performance, [4]. Thus, frameworks that support run-time system designing need to take into account such parameters. Some academic approaches exploit these characteristics such as [1].

## 3. FRAMEWORK DESCRIPTION

Figure 1 depicts the structure of the framework for designing reconfigurable systems. The framework is basically composed of two stages: the Scheduler and the SystemC Reconfiguration Model. Initially, the analysis tools determine the tasks scheduling. After that, an internal reconfigurable representation is created in order to emulate a real reconfigurable environment. So, this stage represents the real platform for the implementation of the design. Between these two stages a parser tool provides a link in order to translate the representation of the scheduling stage for the platform representation in second stage.

The first step in the framework is represented by a particular application. This application is composed of several real-time hardware tasks. Each task of the application is translated in a Timed Petri Net (TPN) internal model. Additionally, a TPN precedence model that represents the data dependence between tasks is applied. Considering the task model and precedence model, a final TPN model is built in order to represent the application.

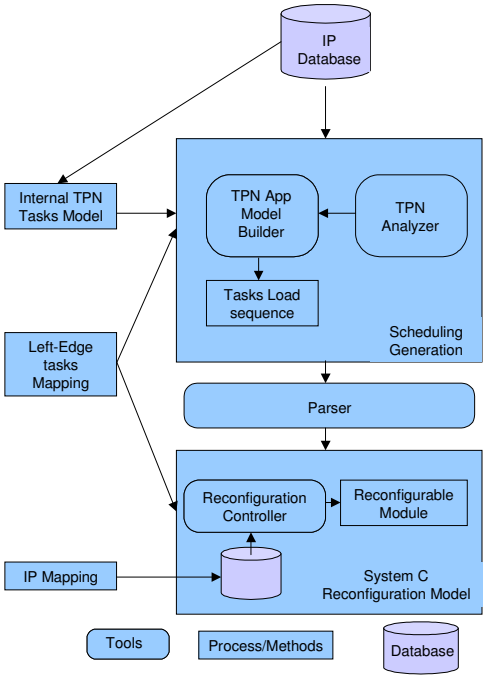


**Figure 1. Framework for Run-time reconfigurable designing**

Petri Nets is used as internal language due to the easiness in representation of parallel and timing events that are strong characteristics of real-time embedded systems. A tool called PNBuilder builds this application model in an internal format. A brief definition of the task model can be placed in the following form:

### 3.1 Tasks Model Definitions

Let $T_{mp} = (P, T, F, W, m_0, \Gamma)$ be a Timed Petri Net model which represents a particular real-time task *Ti*, where *P* represents an ordered set of places, *T*, represents an ordered set of transitions, *P* and *T* are non-empty disjoint sets and represents the two types of nodes of the graph. $F \subseteq (P \times T) \cup (T \times P)$ represents the edges of the net and the flow relation, $W: F \rightarrow \aleph$ represents the weight of the flow relation, $m_0: P \rightarrow \aleph$, represents the initial marking of the net and $\Gamma: T \rightarrow \aleph$, is a function that maps each transition to a bounded time delay. In this definition, the set of places represents the availability of an operation or processing and the set of transitions represents all the task activities. Each transition $t_i$ presents a time associated delay $\Gamma(t_i)$, which represents the time expended by the task activity *i*. The marking corresponds to the number of tokens in places where $m_i : P \rightarrow \aleph$. Therefore, $m_i (p_j)$, $\forall\ p_j \in P$, is the number of tokens on $p_j$ at marking $m_i$.

### 3.2 Scheduler analysis

The scheduler analysis must be performed through a Petri Net analyzer over the application model. This tool allows to check several system properties, such as, feasible firing schedules, reachability, deadlock freedom, starvation-freedom, boundedness, fairness and others characteristics. Particularly, the characteristic of reachability is very important because it denotes that a scheduling process is possible to attend all the real-time task timing constraints.

After mapping of all tasks in the Petri Net Model, the scheduling is obtained and presents the best performance through the shorter time of execution of the entire application (e.g. execution of all tasks). The execution of all the tasks allows the net to achieve a determined state (marking). The algorithm explores all the tracks to find this marking and choose the track that expends less cycle clocks to achieve it.

Initially, the algorithm detects if the net is feasible for a final marking. If yes, the algorithm constructs the state graph of the net. After this processing, the tool chooses the shortest way regarding the firing of all transitions, spending the least possible time. This algorithm presents a complexity level $2^n$, where n means the number of tasks that should be scheduled in the architecture.

## 4. RECONFIGURABLE SYSTEM MODEL

In this session the second stage of the framework is presented: the implementation of the reconfigurable system executable model. Now, two extensible and configurable templates are implemented in SystemC to model the functionalities of the high-level reconfigurable system designs. The templates are defined as RAC (Reconfigurable Area Container), and RCT (Reconfigurable Controller) respectively

Figure 2 depicts the RAC model. This model represents an abstraction of a reconfigurable area in a programmable hardware device. It can be understood as a container that stores IP candidates for reconfiguration. RAC operates, in each moment, with the functionality of one of their candidate modules, emulating in this way, a reconfigurable area. In its construction, a project constraint that restricts the candidate modules to a same external communication interface was considered. In order to represent this abstraction, the RAC was defined through the inheritance concept. The common I/O interface should be defined in an abstract class, which in the defined model, is represented by the *IPInterface* class. The other classes of the model should extend this abstract class, guaranteeing, thus, a common interface among the modules. In this abstract class, a virtual method was also declared called *process()*, that should be implemented in each of the daughter classes, represented by the RAC and IP classes.

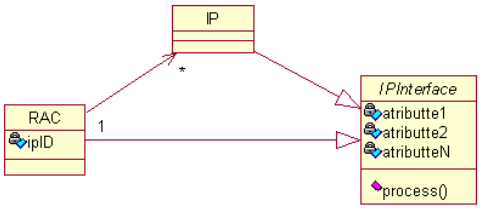


**Fig.ure 2. Reconfigurable Area Container (RAC) model**

Thus, following the proposed model, the implementation should go through the following steps:

1. Design of the Abstract class *IPInterface*
   a. Define the module's common attributes
   b. Declare the virtual method, *process()*
2. Design of the Candidates Modules
   a. Extend the *IPInterface* class through inheritance
   b. Implement the method *process()*, for each defined IP
3. RAC implementation
   a. Extend the *IPInterface* class through inheritance
   b. Instantiate each one of the Candidate Modules
   c. Implement the method *process()*, that should control the modules to be executed through the ipID attribute

The next step is the implementation of a module that can control the reconfigurable areas defined. This function is accomplished by the RCT, presented in figure 3.

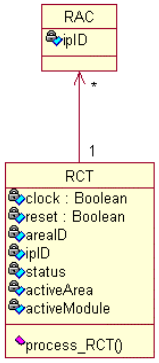


**Fig. 3. Reconfigurable Controller (RCT) model**

In this figure, several RACs are controlled by a single RCT, that is responsible for centralizing all execution and reconfiguration processes of the candidate modules, allocated in the RACs. The idea is to accomplish that control through status information that can store data about the internal state of each managed RAC. As examples of information managed by the controller, we can quote: data from actives modules in each RAC, and which IPs is available for reconfiguration in each area. However, the execution order and the reconfiguration process of the tasks are not defined in this controller. The controller only receives messages through its external interface, and executes the requested operations.

## 5. EXAMPLE APPLICATION

An example that involves the stages described in the framework is detailed. In this example, a test application is composed of 6 well-defined tasks, as shown in table 1 and must be performed in a static scheduling. The tasks present individual deadline timing for execution and the overall application as well. This means that all tasks must be performed before the final timing of the application. The example also presents data dependencies between some tasks.

**Table 1. Test application Tasks**

| Task | Deadline timing (ut) | Data Dependence |
|---|---|---|
| Encrypt Machine | 5 | - |
| State Machine | 10 | - |
| Factorial Generator | 2 | - |
| Gray sequence | 2 | Encrypt Machine |
| Ring sequence | 16 | - |
| ULA | 4 | Factorial Generator, State Machine |
| Application Deadline | 30 | - |

Each task must be mapped in TPN task model. After, all tasks models are joined in a final application model. This final model carries all the timing that is involved in the execution of the application and the data dependence between tasks. Each reconfigurable area must present the same interface as a "virtual slot", in order to enable the reconfiguration of the tasks. After the scheduling generation these slots will be translated in Reconfigurable Area Container (RAC).

Figure 4 depicts the TPN model implemented for this application.

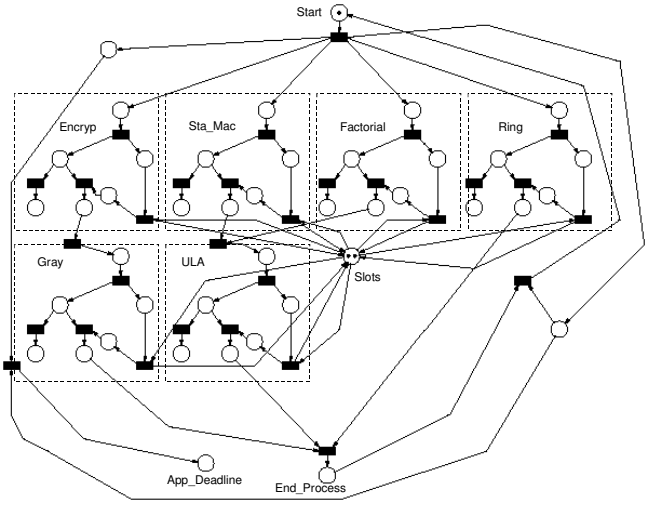


**Figure 4. TPN application model**

When the TPN application model is finished, a PN analyzer tool is used for executing the following steps:

- Search for space state on the net;
- If it's possible a final marking (end Process)
- The way that spends less time in order execute all tasks

Figure 5 depicts the result of this analysis. The diagram shows the best sequence considering two reconfigurable regions (slot1, slot2).

The results of scheduling generation are translated for the high-level reconfigurable system representation in systemC. In this environment, the dynamics of the reconfigurable system can be proved in functionality and efficiency. Figure 6 shows the class diagram that was generated in order to form the reconfigurable controller (RCT) and the Reconfigurable Area Container (RAC) for the application example.

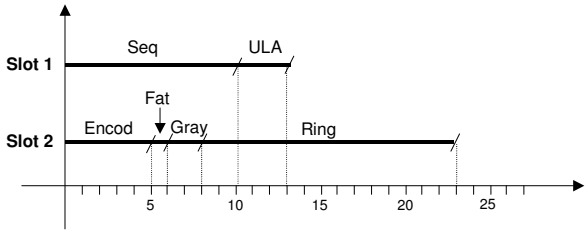


**Figure 5. Scheduling results**

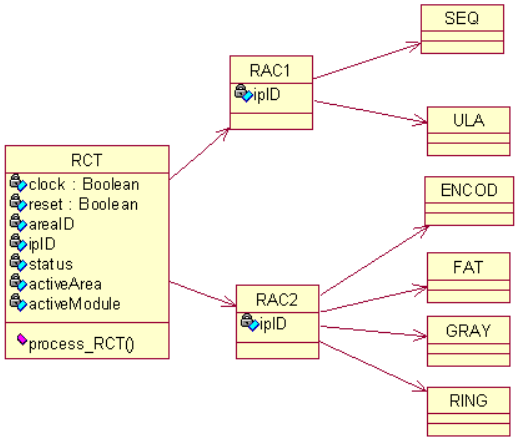


**Figure 6. Class diagram of the application**

## 6. CONCLUSIONS AND FUTURE WORK

In this paper, there was been presented a framework for run-time reconfiguration designing. The framework is split into two major parts: The real-time task scheduling and a high-level representation of reconfigurable environment. Future works consider a refined process of the model presented, in order to enable a synthesizable implementation of reconfigurable system design, and then its implementation in a real reconfigurable platform.

## 7. REFERENCES

[1] H. Walder and M. Platzner, "Reconfigurable Hardware Operating Systems: From Design Concepts to Realizations" in *Proceedings of the 3rd International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA)*, Las Vegas, Nevada, USA, pp 284-287, 2003.

[2] J. Becker, W. Jin and M. Ulmann, "Hardware enhanced Function allocation management in reconfigurable systems," *2005 IEEE IPDPS Int. Parallel & Distributed Processing Symphosium – RAW Reconfigurable Architectures Workshop*, April 2005.

[3] J. Becker, M. Hubner and K. Paulsson, "parallel and flexible multiprocessor system-on-chip for adaptive automotive applications based on Xilinx Microblaze soft-cores," *2005 IEEE IPDPS Int. Parallel & Distributed Processing Symphosium – RAW Reconfigurable Architectures Workshop*, April 2005.

[4] H. Walder and M. Platzner, Online Scheduling for Block-Partitioned Reconfigurable Devices, in *6th Design, Test and Automation in Europe Conference and Exhibition – (Date'03),* Messe Munich, Germany, pp.290-295, 2003