

SyCERS: a SystemC design exploration framework for SoC reconfigurable architecture

Carlo Amicucci Fabrizio Ferrandi Marco Santambrogio Donatella Sciuto

Politecnico di Milano
Dipartimento di Elettronica e Informazione
Via Ponzio 34/5
20133 Milano, Italy

{amicucci,ferrandi,santambrogio,sciuto}@elet.polimi.it

ABSTRACT

Reconfigurable devices, such as FPGAs, introduce into the design workflow of embedded systems a new degree of freedom: the designer can have the system autonomously modify the functionality carried out by the IP-Core according to the application's changing needs while it runs. Increasing the complexity of the design provides to the designers much more flexibility in their decisions but imply that the time to market of the final solution is dramatically increasing to. This paper presents a design methodology for dynamically reconfigurable IP-core based architectures. The methodology is based on the SystemC class library and targets the design of reconfigurable architectures for embedded systems. The use of the SystemC library enables the designer to specify a system in its hardware and software parts with just one language.

1. INTRODUCTION

The possibility of testing dynamic reconfigurable architectures and of validating dynamic reconfigurable systems built on top of these architectures is one of the most relevant research topics in the reconfigurable embedded system design field. Research in this field is, indeed, being driven toward a more thorough exploitation of the reconfiguration capabilities of such devices, so as to take advantage of them not only at compile-time, but also at run-time, which allows the reconfigurable device to be reprogrammed without the rest of the system having to stop running. According to this observation it is clear how the possibility of simulating dynamic reconfigurable architectures and of validating dynamic reconfigurable systems built on top of these architectures is one of the most relevant research topics in the reconfigurable embedded system design field. Among these architectures some are built on top of commercial FPGAs. Reconfiguration of FPGAs can be performed at two levels: register transfer level (RTL) reconfiguration or IP-Core reconfiguration.

Aim of this paper is to propose a methodology that has been implemented into a framework, called SyCERS, based on SystemC, that can be used by system designers, to study and verify their own reconfigurable system specifications. The proposed framework can be used to explore the solution domain in a very fast and flexible for fast design exploration.

A good simulation technique for dynamic reconfigurable

architectures should be based on one of the available hardware description languages (HDL). Moreover this technique has to be flexible, to be architecture independent, and enough scalable, so that it can be used at different levels of abstraction. Finally, the system model, built with this technique, should be executable in a commercial HDL simulator, e.g. Modelsim. The SyCERS framework, is built on top of the SystemC library and allow the specification of both architecture and system models. Classes implementing architectures have to be derived from a set of common interfaces provided in the framework. In addition, an architecture is built as a C++ library, so it can be imported in any system design. The system models use the interfaces, implemented in the architecture to access the hardware resources; therefore a model can be tested in any architecture implementing the same interface set. A system model can be also parameterized so, resources like number of available reconfigurable blocks and memory size, can be changed before execution. This parameterized model allows the test of a system on various solutions. Finally, the SystemC model of the system, built with the framework, can be executed in the standard OSCI simulator or even in the ModelSim commercial simulator.

This paper is composed of four main sections. Section 2 will present, not only the description of other works related to the proposed one, but also the background and all the definitions useful to better understand the proposed methodology. In Section 3 we will describe the proposed methodology and the SyCERS framework, implemented to validate the proposed approach while, Section ?? presents the Caronte architecture modeled in the SyCERS framework. In Section 5, are shows some tests results. Finally Section 6 will close this paper providing also some hints on possible future works based on the one proposed here.

2. PREVIOUS WORKS

Robertson and Irvine in [1] present a design flow for partially reconfigurable hardware and a CAD framework for creating partially reconfigurable designs. The functional simulation of the design is done in the early stage of the suggested flow with the DCSim tool. DCSim is a CAD tool that takes in input a VHDL description of the design and a file (RIF) specifying the reconfigurable behaviour and produces a simulation model. This simulation model can be exe-

cuted through a conventional VHDL simulator. The simulation model uses the Dynamic Circuit Switching (DCS) technique to simulate dynamic reconfigurable hardware. This technique, created by Lysaght and Stockwood [2], operates by adding virtual components in the design in order to model the effect of the reconfiguration. Essentially a virtual component acts as a switch between configurations. The reconfiguration controller can be implemented in software or in hardware. Robertson and Irvine use in their design flow DCSCONfig, a tool that can synthesize a customizable VHDL reconfiguration controller from the RIF file. Also the European community RECONF2 [3] project uses the DCS technique in functional simulation. The project aim is to develop the required design environment to be able to efficiently use dynamic reconfigurable FPGAs.

Clock morphing (CM) [4] is another technique proposed to simulate dynamic reconfigurable hardware but does not use virtual components. The simulation of reconfigurable hardware is achieved through the definition of separate clock signals for each configuration. All these clock signals are controlled by a reconfiguration controller. The controller simulates a configuration of a selected module connecting the module's clock to the system clock. That is done for a period of time determined by a configuration schedule. Module's clocks in the virtual state are set to a special value V (virtual). The V-value propagates from clock input to module outputs.

[5] proposes a system level modeling methodology based on SystemC. According to this methodology all the configurations of a single module were implemented as a functionality of this module. This module implements also the context scheduler, which is responsible of switching between the configurations. In this approach, the possible configurations, are coded inside the architecture description. We present a different approach based on C++ function pointers to pass the configuration at the reconfigurable hardware model. In this way it is possible to describe separately the architecture and the system targeted for the architecture. Our approach can be also used at a high level of abstraction, so it is possible to think at evaluate system configurations since from the early design stage. We can initially define an untimed functional configuration model and refine it into a register transfer level (RTL) model. Our method is also independent from of the specific hardware used, and can be used on FPGA-based architectures or on architectures based on other hardware devices.

In this work we propose a complete simulation flow, based on SystemC, that has been validated using the Caronte architecture, [6], where the use of TLM enables a light simulation model and the use of it in the early phases on the design.

2.1 TLM and SystemC for reconfigurable architecture description

Transaction level modeling [7] is a top-down approach to system design widely adopted in the verification phase of the design. In TLM the system is first described at an high level of abstraction where communication details are hidden, then the description is refined with the necessary details. The key concept of TLM is the separation of functionality definitions from communication details, to achieve this TLM uses through the concept of channel. A channel allows communication through methods calling. Then, two functional

components, interconnected with a channel, communicate calling the methods exported by the channel. In this way the communication detail is hidden in the channel definition. SystemC enables TLM modeling through `sc_interface`, `sc_channel`, and `sc_port`. A channel interface can be derived from `sc_interface`, then the `sc_interface` can be implemented in a `sc_channel`. At this point a `sc_module` can communicate through a `sc_channel`, using a `sc_port` connected to the `sc_channel`. The OSCI is also developing a TLM standard library.

3. THE PROPOSED APPROACH

A dynamic reconfigurable architecture often needs software integration to control the scheduling of the reconfiguration. SystemC enables to co-develop software and hardware, so it is possible to describe a complete hardware/software system with just one language. JHDL is an Hardware Description Language implemented with the Java language and is designed for implementing reconfigurable architectures on FPGA. In the JHDL simulation kernel it is necessary to stop the clock to add or delete circuits. This is unacceptable if the architecture to model is partially reconfigurable, since it is not possible to time-modeling the operation of reconfiguring part of the system while the rest is still running. With SystemC and our approach, partially reconfigurable systems can be easily modeled at TLM level with simple method calling.

The developed framework, SyCERS, supplies a set of interfaces to describe a dynamic reconfigurable architecture. The framework core supplies the interfaces and the reconfiguration controller implementation that can be plugged via interface to any dynamic reconfigurable architecture implementation to control reconfiguration at run-time. The main idea, described in 3.2, is the definition of a methodology to implement a dynamic reconfigurable system using SystemC. Those systems can be built both on top of a library containing the descriptions of a set of reconfigurable architectures, or allowing the system application designers to define their own architectures and scheduling policies.

3.1 Modeling Dynamic Reconfigurable Architectures with TLM

Using a set of interfaces between the application and the architecture model is useful to maximize architecture descriptions and application specifications re-usability. In this scenario the designers of reconfigurable architectures have to implement the description of their architectures as C++ libraries, deriving the classes for those descriptions from the core interfaces, defined in the SyCERS simulation framework. That is exactly the same situation for the application developers, therefore they can use the same set of interfaces to access the desired architectures resources. This also means that if two architectures implement the same interfaces an application can be tested on both architectures without any additional work.

All the interfaces, involved in the reconfiguration processes are represented in Figure 1. The interfaces proposed in the SyCERS simulation framework to model dynamic reconfigurable architectures are mainly two and the `sc_module` used to model a dynamic reconfigurable hardware component has to implement both:

- The `rec_hw_control_if` is used by the reconfiguration

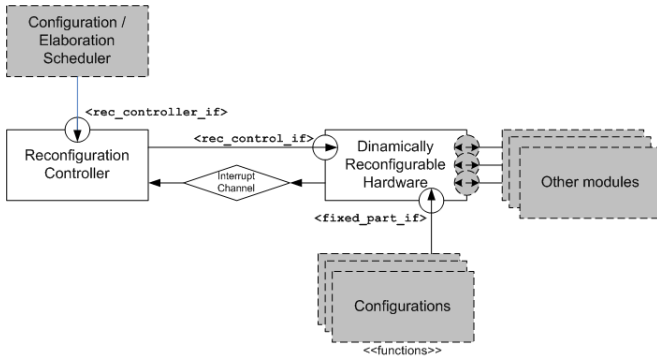


Figure 1: Reconfiguration control interfaces

controller to control the reconfigurable component resources. The configuration is passed as a function pointer as explained in section 3.2 and a calling to the elaborate method starts the elaboration while the erase method will finally erase the present configuration.

- The `fixed_part_if` is used by the dynamic components to allow the correct binding between a configuration, described in 3.2, and its *physical shell* defined in the reconfigurable architecture.

There is also a third interface which is given to plug into the simulation framework a new policy for the reconfiguration scheduler within the reconfiguration controller provided with the framework. This scheduler is used to implement the desired policy to correctly schedule all the reconfigurations and the computations on the chosen reconfigurable architecture. All the proposed interfaces are TLM-based but architecture and application models can be implemented at any abstraction level using the same auxiliary adapter class as in [8].

3.2 Modeling Dynamic Reconfigurable Processing Elements with SystemC

In the proposed framework hardware component has been described, using the SystemC description language, as a C++ class derived from the `sc_module` class. Due to the fact that no `sc_module` object, neither any objects derived from this class, can be initialized after the beginning of the simulation process, it is not possible to model a module reconfiguration process as a new `sc_module` initialization during the simulation phase. The approach followed in the SyCERS framework to model the system reconfiguration using SystemC is based on the simple consideration that the functionalities of an `sc_module` could be implemented as `SC_METHODs` and, or `SC_THREADs`, therefore using one of them as a wrapper to call a function pointer we are allowed to change the functionality of modules varying only this function pointer, as shown in Figure 2.

Under this assumption the SyCERS framework can model the reconfiguration process using a function pointer, an `sc_mutex` and just one execution thread for the reconfigurable component. The reconfiguration process, adopted in the SyCERS framework, is shown in Figure 3 and it can be described by the following phases:

- **Locking Phase:** During this phase the framework re-

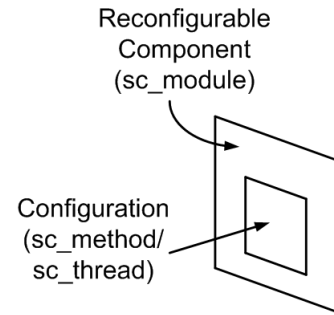


Figure 2: SyCERS Reconfigurable Component Overview

acts to a reconfiguration process that has been invoked by the reconfiguration controller. The access to the module that has to be reconfigured is disabled locking its `sc_mutex` for the time necessary to the reconfigure process to complete.

- **Configuration Phase:** After blocking the access to the module that is going to be reconfigured, its function pointer is set to the new function, the new configuration, that has to be implemented by the module. Once the new pointer has been set, the mutex is unlocked.
- **Elaboration Phase:** After the first two phases the module is ready for its computation. During its elaboration, the computation, initiated as in the reconfiguration case, by the controller, the mutex of the module is locked a second time to prevent an unwanted events. Once the module elaboration computation is completed the mutex in the module is unlocked again.

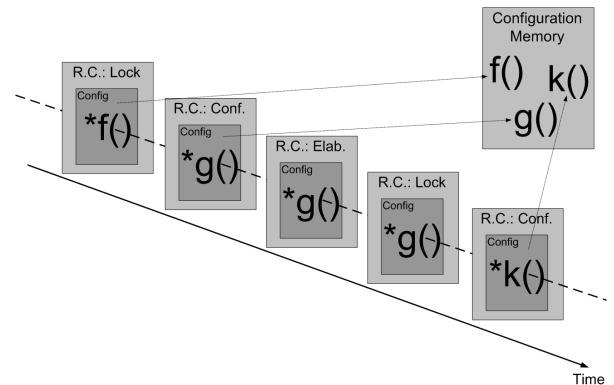


Figure 3: SyCERS reconfigurable process

Since the configuration and elaboration processes are mutual exclusive it is also possible to use just one `SC_THREAD` to model a reconfigurable hardware, so that the number of threads in the SystemC model is minimized and the simulation is faster.

4. THE PROPOSED ARCHITECTURE

This section shows the architecture, which is going to be used as the base architecture that has to be described and

simulated into the proposed simulation framework, as presented in Section 4.3. The core of the architecture is the PPC405 processor which implements both the controller and the scheduler of the given system implementation. Figure 4 shows the complete architecture stressing its two sides: the fixed one and the reconfigurable one.

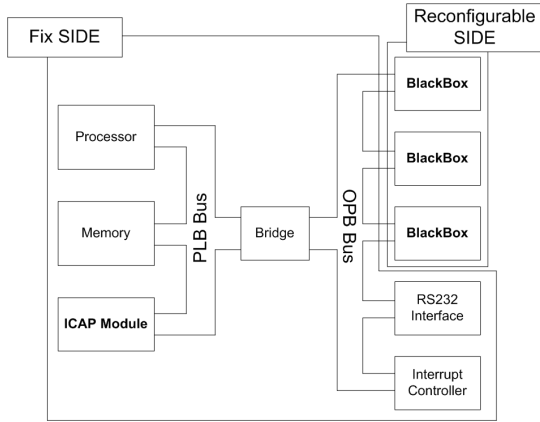


Figure 4: Caronte Architecture Overview

4.1 The Fixed Architecture of Caronte

4.1.1 The Body

The body of the Caronte architecture, [9, 10], is the real physical implementation of the fixed part. It is basically a Von Neumann architecture composed of five classes of components:

- **ICAP**, used to read/write a configuration from/to the BRAM to/from a specific BlackBox;
- **Memory**, used to store all the partial bitstream data information;
- **Buses**, used to implement the architectural communication infrastructure. It is possible to identify two different kind of busses:
 - The *IBM CoreConnect technology*, that represents the 90% of the entire communication system of the architecture;
 - The *bus macro technology*, which provides a fixed bus of inter-design communication. Each time partial reconfiguration is performed, the bus macro is used to establish unchanging routing channels between modules, guaranteeing correct connections.
- **PPC405 Processor**, used to provide the physical support for the *mind*;
- **Interrupt Controller**, used by the PPC405 processor and the BlackBoxes to dialog one to each other.

4.1.2 The Mind

The Caronte Mind is the software part of the architecture. In a previous implementation, the Caronte software is implemented as a *standalone* system while now it has been based on a Linux operating system. Caronte implements an

architecture where, at the right time, each processing element can be mapped, thanks to the controller, according to the placement information. The time of the reconfiguration has been computed statically, but it can be modified due to the actual execution or external environment inputs. This is the reason why the PPC405 runs also a dynamic scheduler, which takes into consideration the runtime implementation of the system description. Caronte sees each processing element as a component that has to be mapped onto a specific FPGA area.

4.2 The Reconfigurable Side of Caronte: Black-Box Definition

A BlackBox is a fixed known portion of the FPGA that can be completely reconfigured without interfering with the execution of the remaining part of the FPGA. Therefore, a BlackBox can be considered as a shell for processing elements. This shell provides also the communication channel interface between the node and the system. This interface allows the node to send data directly on the communication channel or to temporarily store a fixed number of data in its communication spooler, a data repository used during the reconfiguration action. A BlackBox can be considered as a virtual shell used to contain different processing elements of the system description. In order to be able to implement a partial reconfiguration of a portion of the FPGA it is important to know which is the portion that has to be reconfigured. The Xilinx Platform Studio Tool of EDK, used to create FPGAs architectures, offers an automatic synthesis engine that generates a real project implementation by arranging each logic unit in a standard way. A BlackBox provides the interfaces needed by the VHDL description of a processing elements to dialog with all the other components of the architecture, such as the CoreConnect bus, the processor, the interrupt controller and the other BlackBoxes.

During reconfiguration the Processing Element node logic will be modified, while the communication interface and IP Interconnect (IPIC) between the node logic and the interface will remain the same. This means that a BlackBox is constituted by two VHDL, Verilog or EDIF files, the first one containing the *architecture-dependent* logic interface and the second one the processing element hardware description.

4.3 The simulation model

The simulation model was developed using the *Open SystemC PowerPC core models* from IBM. This is a library of SystemC cores that models the *IBM CoreConnect* open standard bus architecture. This library contains also a *standalone PowerPC Instruction Set Simulator (ISS)* which can be connected to the *Processor Local Bus (PLB)* SystemC model. Using this library it has been possible to design and co-simulate the hardware and the software part of a SoC based on the CoreConnect bus architecture, in this case the Caronte architecture. The software part is executed on the ISS while the hardware is modeled as a set of SystemC cores connected to the OPB bus or the PLB bus.

4.3.1 Simulation flow

Figure 5 shows the Caronte simulation flow.

To build an application for the SyCERS framework using the Caronte architecture model it is necessary to perform the following steps:

- **Control Description Phase:** To be able to execute

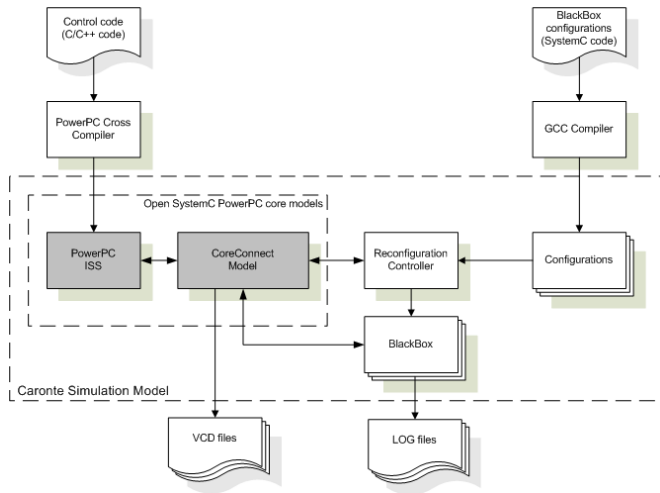


Figure 5: Caronte simulation flow

the application onto the Caronte architecture, not only in the SyCERS framework but also in the actual platform, the designer of the application has to specify the code that will represent the controller of the system. The only difference between the real Caronte and its SyCERS model is that the code, in the second case, will be used with the PowerPC ISS, instead of the real PowerPC available onto the FPGA. This phase can be easily completed by writing the C or C++ code and cross compile it for the PowerPC.

- **BlackBoxes Setup Phase:** In this phase it is necessary to assign the correct number of available BlackBoxes to the Caronte model that the designers want to use to run its application. Those BlackBoxes, once defined, have to be correctly bound to the reconfiguration controller core.
- **BlackBoxes Design Phase:** This third phase can also be named as *Reconfiguration Design Phase*, and it is used to define all the configurations necessary to describe the application. All these reconfigurations are considered as functions that will be assigned at the correct moment to a specific BlackBox, as described in section 3.2.
- **Caronte Setup Phase:** Finally, after the definition of the available resources, phase two, and the descriptions of all the necessary functionalities, the third phase, the designer is able to set-up its own customized version of the Caronte architecture to be simulated using the SyCERS framework. This phase consists only in the compilation of all the necessary description files for the machine on which the SyCERS framework has to be used.

Let us see in detail, in the following section 4.3.2, how the BlackBoxes and the reconfiguration controller are modeled using SystemC.

4.3.2 BlackBox and reconfiguration controller SystemC models

BlackBoxes are the dynamically reconfigurable components used in the Caronte architecture. They are defined as

sc_modules connected to the on-chip peripheral bus (OPB) and they can both be used masters or slaves on the bus. Both those behaviours have been modeled: being master means that the BlackBox will use a sc_port versus the OPB bus, while, in the slave mode, it has to implement a sc_interface that is used by the OPB, to read and write the BlackBox registers. To implement the visibility of its set of registers the BlackBox implements the *BlackBox communication interface* (bb_com_if), which is characterized by four methods:

- **bool write(address, data):**
This method is used to write desired data, via the OPB bus, to the BlackBox.
- **bool read(address, data, width):**
The read method implements the read mechanism that can be used to read the value of the BlackBox internal register. If the address provided as input to the read or write method is in the BlackBox address range, the required operation will be performed on the internal register otherwise the operation will be dropped.
- **address get_blackbox_base_addr():**
This method is used to setup the correct BlackBox base address.
- **address get_blackbox_high_addr():**
This last method is very similar to the previous one and it is used to assign the correct high address used to define, with the base address, the address space for a BlackBox.

The BlackBox implements also the rec_ctrl_if, used by the reconfiguration controller to manage the reconfiguration properly. Therefore, the reconfiguration controller locks the BlackBox for the necessary simulation time calling the *configure()* method in the rec_ctrl_if. Both the bb_com_if and the rec_ctrl_if are specified at TLM level, thus hiding all the communication details to the configuration developer. This provides a speedup of the development time and permits to use the simulator in the early stages of the system design. The reconfiguration controller is an OPB slave directly connected to each BlackBox. The reconfiguration controller passes the function pointer implementing the configurations to the BlackBoxes models and locks them for the necessary reconfiguration time.

5. TESTS AND RESULTS

The following paragraphs will present two different systems simulated using the Caronte architecture, [9,10], modeled in the proposed framework.

5.0.3 MD5

The system takes in input a file of arbitrary length and produces as output a 128-bit digest of the input file using the MD5 algorithm. The MD5 algorithm is intended for digital signature applications, where a large file must be "compressed" in a secure manner before being encrypted. The system was implemented partitioning manually the MD5 C code contained in the RFC1321.

In MD5 the input message is broken up into chunks of 512-bit blocks. The main algorithm then works on each 512-bit block in turn. The processing of a message block consists of four rounds, each block modifies an intermediate state.

No parallelization is possible because the intermediate state in one iteration is built on the intermediate state of the previous iteration.

The four rounds have been chosen to be implemented as reconfigurations. According with the *natural* system partitioning the maximum number of reconfigurable resources necessary are four, one for each round. Also other architectural solution have been tested. The difference between these solutions consists in the number of BlackBoxes available in the Caronte architecture. Table 1 shows the execution time of a 512-bit block elaboration on a system with only one BlackBox, that means that it is necessary to reconfigure before each round, with a fixed reconfiguration time of 40 ms.

Table 1: MD5 on a one BlackBox

Operations	Execution time (ms)
Reconfiguration (round_1)	40
Elaboration (round_1)	0,00144
Reconfiguration (round_2)	40
Elaboration (round_2)	0,00144
Reconfiguration (round_3)	40
Elaboration (round_3)	0,00144
Reconfiguration (round_4)	40
Elaboration (round_4)	0,00168
Tot.	160,006
Reconfiguration time	160
Elaboration time	0,006

It can be seen that the system spent 99.9% of the execution time reconfiguring and just 0.1% of time doing an useful elaboration. From the test results can be seen that the reconfiguration time is one magnitude order bigger than the execution time so implementing MD5 on one BlackBox could be considered, at a first view, unacceptable.

This example shows how the simulator can be used to find out the best *behaviour* of the architecture to justify a reconfigurable solution. Due to the results proposed in Table 1 the designer can decide to adapt the execution model to be able to justify the reconfiguration approach using a model similar to the one proposed in [11]. The idea is to iterate the execution of each block on a sufficient amount of data to enlarge its computation time until the execution time is bigger than the one needed to reconfigure the BlackBox, this model as shown in Figure 6.

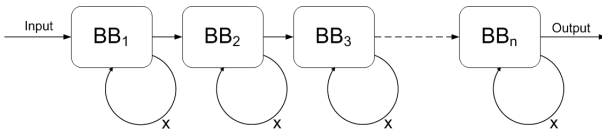


Figure 6: Execution model.

The *X*value, computed according to the specif situation, represents the number of iterations that are computed on a generic processing element that makes its computation on a set of data of a fixed size each time. According to

this observation the new version of the Caronte architecture will be tested using data intensive algorithms such as those belonging to the computer vision area, see section 5.0.4.

5.0.4 Canny filter

To test concurrent access to hardware resources an image processing algorithm, running concurrently several instances of them varying reconfigurable hardware resources on the architecture model, has been implemented. The algorithm tested is an implementation of the Canny edge detection algorithm. It has been designed using four configurations: a Gaussian filter, a Canny Enhancer, a non maximum suppression block and an hysteresis thresholding block.

For each image in input the framework create a process using SystemC facilities. Each process try to reserve the necessary memory to compute the assigned image. After memory the reservation phase the necessary elaboration sequence will take place.

The graph in Figure 7 shows the number of operations, elaborations or reconfigurations, for an architecture with one BlackBox.

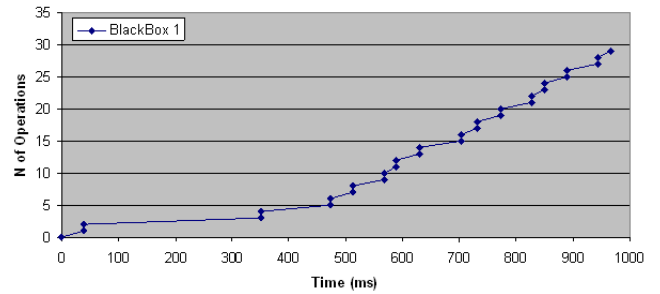


Figure 7: The Caronte Architecture - One BlackBox solution

With such a solution its necessary to introduce a reconfiguration after each computation, as shown in Figure 8, to be able to implement the Canny algorithm, which has been characterized using 4 configurations, onto the Caronte architecture.

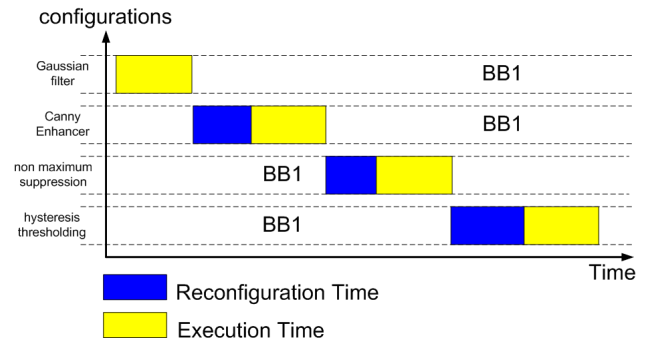


Figure 8: Canny simulation - One BlacBbox solution

In Figure 9 is reported a comparison between different architectural solution.

Using the proposed framework is easy to show that increasing the number of reconfigurable elements does not imply better performances, a shorter execution time, of the

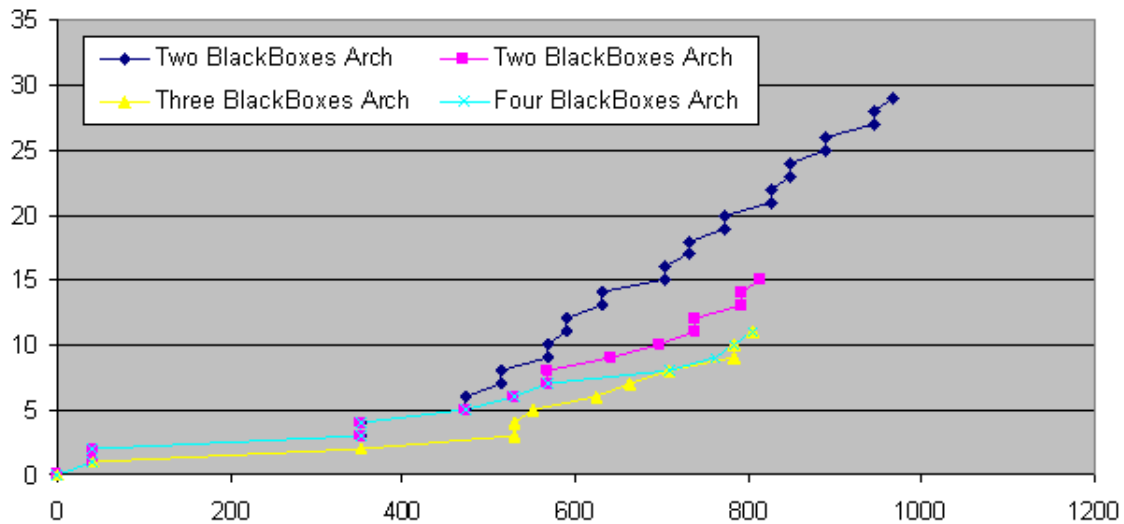


Figure 9: Comparison between the four architectures

system. As clearly shown in Figure 9 it is not necessary to have more than two reconfigurable BlackBoxes to reach the best implementation, but without this information provided by the proposed framework, it is not so difficult to implement a solution with the wrong number of reconfigurable elements that will produce a more complex schedule without any improvement in the final solution.

6. CONCLUSIONS

Preliminary results show that the SyCERS framework, based on the SystemC language and the Transaction Level Modeling, combined together to describe reconfigurable architectures, enable the exploitation of all the benefits of a reconfigurable architecture. The tests have been implemented, with success, trying to figure out different characteristics of the SyCERS framework, such as the ability of helping the designer in choosing the correct number of reconfigurable elements to design his/hes system.

7. REFERENCES

- [1] I. Robertson and J. Irvine. A design flow for partially reconfigurable hardware. In *ACM Transactions on Embedded Computing Systems, Vol. 3, No. 2, May 2004*, 2004.
- [2] J. Stockwood and P. Lysaght. A simulation tool for dynamically reconfigurable field programmable gate arrays. In *IEEE Transactions on VLSI systems, Vol. 4, No. 3, September 1996*, 1996.
- [3] G. Habay P. Butel and A. Rachet. Managing partial dynamic reconfiguration in virtex-ii pro fpgas. In *Xilinx Xcell Journal, Fall 2004*, 2004.
- [4] M. Vasilko. Improving simulation accuracy in design methodologies for dynamically reconfigurable logic systems. In *IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'99), Napa, CA, USA, April 21-23, 1999*, 1999.
- [5] K. Masselos A. Pelkonen. System-level modelling of dynamically reconfigurable hardware with systemc. In *The 10th Reconfigurable Architectures Workshop (RAW 2003), Nice, France, April 22, 2003*, 2003.
- [6] Alberto Donato, Fabrizio Ferrandi, Massimo Redaelli, Marco D. Santambrogio, and Donatella Sciuto. Caronte: a complete methodology to implement partially dynamically self-reconfiguring embedded systems on modern fpga. In *IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM 2005)*, 2005.
- [7] L. Cai and D. Gajski. Transaction level modeling: An overview. In *CODES+ISSS'03, October 13, 2003, Newport Beach, California, USA*, 2003.
- [8] J. Pierce A. Rose, S. Swan and J. Fernandez. Transaction level modelling in systemc. In <http://www.systemc.org/>, 2005.
- [9] Fabrizio Ferrandi, Marco D. Santambrogio, and Donatella Sciuto. A design methodology for dynamic reconfiguration: The caronte architecture. In *The 12th Reconfigurable Architectures Workshop (RAW 2005)*, 2005.
- [10] Alberto Donato, Fabrizio Ferrandi, Marco D. Santambrogio, and Donatella Sciuto. Exploiting partial dynamic reconfiguration for soc design of complex application on fpga platforms. In *IFIP VLSI-SOC 2005*, 2005.
- [11] R. Maestra, F.J. Kurdahi, M. Fernandez, R. Hermida, N. Bagherzadeh, and H. Singh. A framework for reconfigurable computing: Task scheduling and context management. *IEEE Transaction on Very Large Scale Integration (VLSI) Systems*, 9(6):858-873, December 2001.