FRONT-END TOOLS FOR DYNAMIC RECONFIGURATION IN FPGA DEVICES

K. KEDZIERSKI, J.M.MORENO-AROSTEGUI, J.CABESTANY
TECHNICAL UNIVERSITY OF CATALONIA, SPAIN

KEYWORDS: FPGA, Dynamic Reconfiguration

ABSTRACT: CAD environment for dynamic reconfiguration in FPGA devices has been depicted. The environment is co-operating with the HDL (Hardware Description Language) tools currently available on the market. It leads a designer from an input project division into a number of partitions till a synthesis process. The environment is a collection of afew applications that form the basic interface between a designer and a target device. All of the applications, that the environment includes, have been described in the paper, together with the design flow example.

INTRODUCTION

Although the possibility of implementing dynamically reconfigurable logics exists, there is still a long way to go in order to achieve standard design techniques for these devices. This paper reports on the work that has been done in order to create an environment, which would lead the user to the final reconfiguration process.
The Front-End Tools, *FET*, form the basic interface between the designer and the FPGA device that is going to use the dynamic reconfiguration. These CAD tools are device independent, which means that they produce that may be used by any device. To achieve that, FET's outputs should be next loaded to the Back-End Tools, responsible for the final preparation of the configuration bitstreams.

GLOBAL VIEW

To be able to understand the Front-End Tools characteristic, one has to concentrate on the following terms:
Dynamic reconfigurable FPGA is a field-programmable gate array with a capability of changing the behaviour of one part of its logic infrastructure while the rest is running.
The static part is a part of an input design that is active during the whole application runtime. It is placed in the "static" area of a target device that is kept intact all the time. In addition to its standard function, it has to provide an infrastructure to load and unload other (dynamic) parts of the design, which is system scheduling, data management, and interface management.
The dynamic parts (dynamic modules – d_modules) are independent parts of the input design that need not be active during the whole application runtime. They share common areas (slots) inside a target device; this is based on the assumption that they are not required to run at the same time in parallel. They are loaded to and unloaded from a target device as requested by the system scheduler.
DCF file (Dynamic Constraint Format file, also called constraint file) specifies the conditions under which dynamic modules are loaded and unloaded to a FPGA device. Moreover, this file describes data and interface management characteristics.
Data management takes care of internal states of d_modules. It saves all marked signals before a module is unloaded and it restores them whenever the corresponding d_module is loaded again. The designer must specify these signals in the constraint file.
Interface management handles interfacing between different dynamic parts and the static part. It holds the last values of interface lines when the dynamic modules that generated them are removed. The designer must specify these interface lines in the constraint file. From the FET point of view data and interface management is done by means of the CUT entities – units built of registers.

Design environment

Fig 1 presents a simplified view of the *FET* design environment, where black squares signify tasks specific to partial dynamic reconfiguration and requiring dedicated tools – FET; ellipses signify files; squares with rounded corners signify tasks identical to the ones of the classical design flows, using "standard" tools.
At the beginning, the designer is able to specify the *DCF* file as the *FET's* input together with synthesisable, static VHDL code, which is the description of the design. DCF file is the description of the constraints and it has been precisely described in [4].
As it may be observed in the Fig 1, the following Front-End Tools may be distinguished:
Automatic Partitioning Tool. The tool relieves the user in looking for the optimal partitioning solution. It performs the detailed analysis of the individual concurrent statements of the input VHDL code. It estimates the resources required for the implementation of these statements and extracts the time dependencies among them. Using this information the tool proposes to the user the optimal dynamic implementation of the partitioned system. The tool may be also used via the

Manual Partitioning Tool. Automatic Partitioning Tool has been precisely described in [7] and [8].

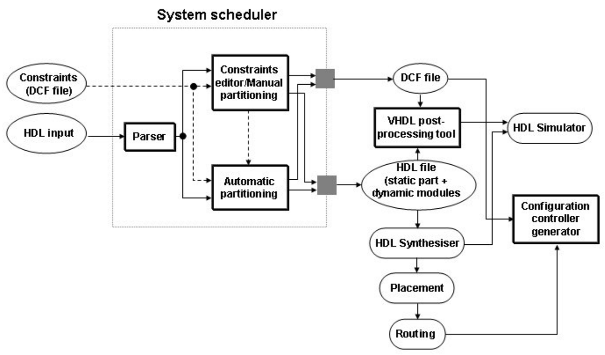
System scheduler
Constraints (DCF file)
HDL input
Parser
Constraints editor/Manual partitioning
Automatic partitioning
DCF file
VHDL post-processing tool
HDL file (static part + dynamic modules
HDL Simulator
Configuration controller generator
HDL Synthesiser
Placement
Routing

Fig 1 Front-End Tools design environment, simplified view.

Manual Partitioning Tool. The tool allows the user to undertake the manual partitioning, also called the constraints-driven partitioning, since the tool is able to read the *DCF* file and to modify the design according to the information found in this file. At the beginning, the user loads the set of VHDL files. These files are initially static. Now the user is able to decide which parts of the code will become dynamic – they will be assigned to Dynamic Modules. Moreover, a number of constraints describing dynamic modules behaviour is attached to the design by the user – at the end these constraints will be saved as the *DCF* file. Therefore, this tool enables to describe all the functionality from the dynamic reconfiguration point of view.

VHDL Post-Processing Tool. The tool permits to carry out a functional simulation of the dynamic description resulting from a manual or automatic partition using standard VHDL simulators. For the user convenience, it has been built as a part of the Configuration Controller Generator.

Configuration Controller Generator. The tool is able to generate C code that may be later compiled in order to provide a software controller for the dynamic implementation of the system. Moreover, it is able to generate the hardware, synthesisable VHDL description of the configuration controller. At the end, the generated VHDL files may be merged together in the top-level file for the functional simulation (with the cooperation with VHDL Post-Processing Tool) or for the hardware implementation of the system.

VHDL Parser. There are two parsers implemented in the Automatic Partitioning Tool and Manual Partitioning Tool. In both cases, these parsers are responsible for converting the input VHDL files into the Internal Representation Format, *IRF*.

SYSTEM SCHEDULER

There are three possible ways of partitioning a design for its implementation in dynamically reconfigurable FPGA, depicted in the Fig 2:

Constraints-driven partitioning. This is done by mean of the Manual Partitioning Tool. Here is where the designer has most control over the partitioning process, being able to define what part of the design will remain static, all the dynamic modules that will be present and the constraints that will condition the loading or unloading of them.

Automatic partitioning. This is done by means of the Automatic Partitioning Tool. This method has as its input a static design that will be partitioned into area-balanced, time-independent partitions that will be executed sequentially in a dynamically reconfigurable FPGA.

As it may be observed in the Fig 2, netlist converter is the first entity that reads the input VHDL files in this method. It is responsible for converting the input set of files into a subset of VHDL containing only processes with sensitivity list, to ease-up the task of the partitioning algorithm. These processes are considered by the algorithm as the elementary units that may be moved between different partitions.

The area estimation of the VHDL processes from the intermediate netlist format constitutes a fundamental input for the partitioning algorithm, as what it primarily tries to achieve are area-balanced and time-independent partitions, or constrained size partitions (by mean of the input *DCF* file).

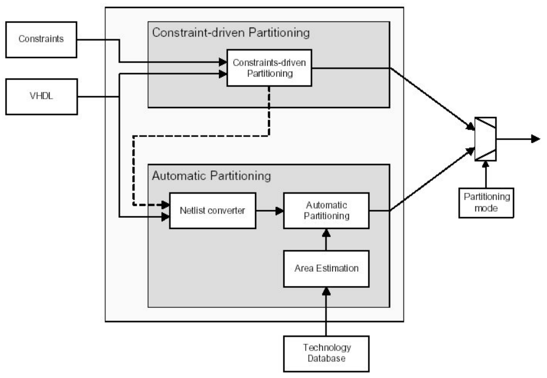

Fig 2 System scheduling detailed view

Semi-automatic partitioning. This is the intermediate mode between the two methods introduced previously. The designer may start the automatic partitioning and next modify it in the constraints-drive partitioning mode, since Manual Partitioning Tool allows the user to load the DCF file together with importing the external dynamic modules. In the Fig 2, this is depicted by means of the dashed line.

Manual Partitioning Tool

Fig 3 depicts general view of the constraints editor with its main parts. The external inputs to the constraints editor are a static VHDL description, and a set of constraints – DCF file. The static description accepted by the tool is a subset of VHDL code that is defined in IEEE standard 1076.6. The designer needs to remember that for all libraries used within the input VHDL description, except for those defined by the IEEE, the source files must be made available.

DCF Parser. The purpose of this entity is to process the input DCF file, checking for errors and creating a list of commands that are used later on the process chain by the partitioner. Since loading the DCF file to the tool is optional and partitioning algorithm is using the DCF parser as well, DCF parser includes the DCF editor. The task of this block is to allow the user to add the constraints for the dynamic modules that are finally going to be saved in the output DCF file. It is important to distinguish these two DCF files: the input DCF file is the base (optional) for the output one. The output one will be used by next FET applications in the design flow, since it contains information specific for the dynamic reconfiguration.

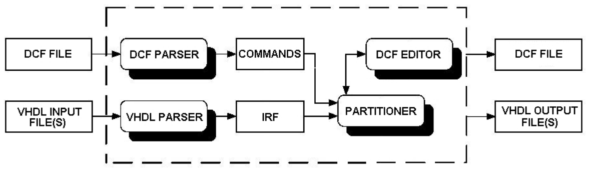

Fig 3 Constraints editor general view. Shaded boxes represent tasks while the other boxes represent some form of data: inputs, outputs or intermediate results.

VHDL Parser. The task of the VHDL parser is to process the input files representing the VHDL description and to convert it into an Internal Representation Format. An important assumption at this stage is that the VHDL code is correct. Neither syntactic nor semantic checks are performed here.

DCF Editor. This block allows both the designer and the manual partitioner to describe the dynamic behaviour of the project by the mean of the *DCF* constraints. These constraints are going to be finally saved in the output DCF file.

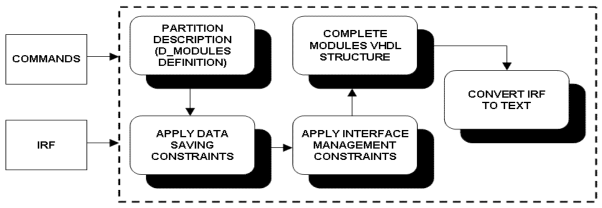
COMMANDS
IRF
PARTITION DESCRIPTION (D_MODULES DEFINITION)
COMPLETE MODULES VHDL STRUCTURE
CONVERT IRF TO TEXT
APPLY DATA SAVING CONSTRAINTS
APPLY INTERFACE MANAGEMENT CONSTRAINTS

Fig 4 Detailed view of the partitioner. Shaded boxes represent tasks while the other boxes represent inputs.

Partitioner. This is the core of the application. The task of this block is to apply the commands from the DCF parser to the IRF representation of the input VHDL design, as shown in the Fig 4. Since the user has the possibility of adding the constraints to the input DCF file, which means in fact changing the loading and unloading conditions for the dynamic modules together with their content (modifications of the dynamic reconfiguration information), this block is able to modify the commands from the DCF parser or to create them from the scratch by mean of the cooperation with the DCF Editor. The partitioning task is divided into several sub-tasks. At the beginning, dynamic modules are created. It is worth mentioning that all the VHDL design description that is not included in the dynamic modules is initially static. Data saving constraints are then implemented. It means that special registers are defined in the static part of the design. These flip-flops reflect dynamic module's outputs while they are in active mode, and maintain the last value once they are deactivated. Next, the structure of both the dynamic and static parts of the design needs to be completed to reflect all the modifications done. At the end, output VHDL files are formed.

Tool description. The Manual Partitioning Tool permits the user to enter several VHDL files corresponding to a static description of the design with pointing at the top-level file, which puts all the files together as components. Once the files are parsed, they are represented graphically so that it is possible to observe the overall hierarchy of the design. The tool allows for the definition of different dynamic modules in an output window. Once these dynamic modules are defined, the user may assign part of the static functionality to them just by dragging and dropping objects belonging to the static description into the dynamic modules previously defined. The user may set constraints for the dynamic modules to specify the conditions for loading/unloading functionality into the dynamic reconfigurable FPGA. These conditions may be time based (executed one time), frame based (periodic) or signal event based (asynchronous). Moreover, exclusive constraints between the dynamic modules sharing the same area in the device may be defined here. A file in DCF format containing the constraints set by the user is also generated. Moreover, constraint syntax checking functions have been also designed, so that the user is informed if the constraints defined correspond to the DCF syntax methodology, defined in [[4]]. The tool generates the individual VHDL files corresponding to the static part of the design and to the dynamic modules defined, as well as to the buffer resources (i.e. registers) managing the communication between the dynamic modules and the static part (cut entities).

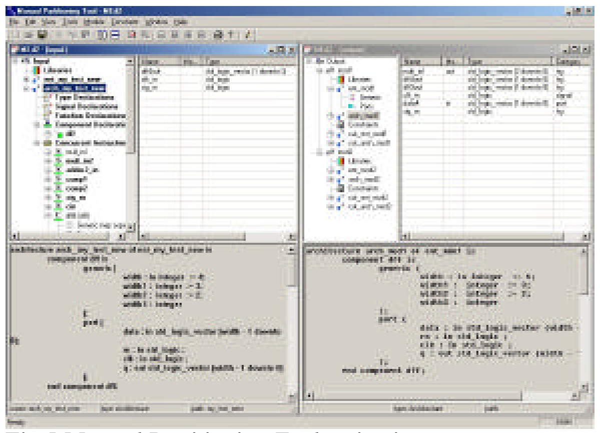

Fig 5 Manual Partitioning Tool main view

There are two main windows in the main application view, as it may be observed in the Fig 5. The one on the left represents the static part of the design, while the one on the right shows the dynamic one. After the files are loaded to the tool, the graphical representation of the VHDL description may be observed in the static part of the design. Therefore, all the design is initially static – the user is specifying which parts of the static description should be moved to the dynamic one. All the action that is required from the designer to divide the initially static VHDL description into a number of dynamic modules is to drag one of the concurrent statements from the static part and drop it on the module icon placed in the dynamic part. All the necessary calculations and corresponding VHDL source code generations are done by the tool.

There is a number of mechanisms implemented in the Manual Partitioning Tool that allows the designer to have the influence on the partitioning algorithm. It is worth to enumerate the “Signal Management” and “Automatic Partitioner”, among others. The first one lets the user to change the default signal interface, while the second one allows using the Automatic Partitioning Tool directly from the application menu functions.

CONFIGURATION CONTROLLER GENERATOR

The Configuration Controller Generator, *CCG*, is a collection of two tools: the VHDL Post-Processing Tool (Functional Block of CCG) and the Configuration Controller Generator core (Technology Dependent Block of CCG). Fig 6 depicts the organization of the CCG. There are two main blocks that the reader should notice, among others: Functional Block, *FB*, and Technology Dependent Block, *TDB*. The first one is in charge of generating the functional description of the configuration controller, while the second one takes care, in addition to what the functional block does, of specific characteristics from dynamic reconfigurable FPGAs. The task of the configuration controller, that is going to be generated by the *CCG*, can be decomposed in the following sub-tasks: detection of reconfiguration conditions (detects of the load and unload conditions for every dynamic module, queues the reconfiguration conditions, according to their appearance) and reconfiguration of the device (applies data saving and interface management constraints, accesses the configuration data from external memory and reconfigure the d_FPGA, apply data restoring).

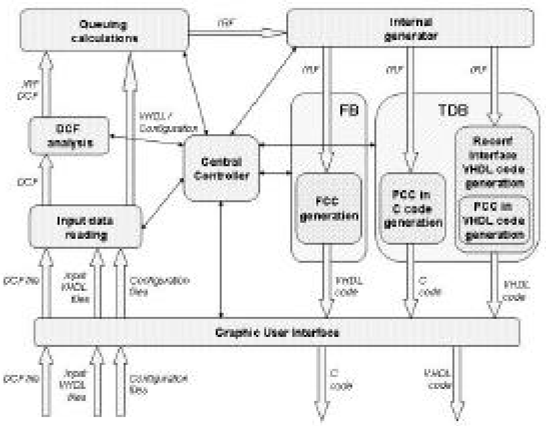

Fig 6 Detailed organization of the Configuration Controller Generator. FB means Functional Block, while TDB means Technology Dependent Block

VHDL Post-Processing Tool – Functional Block

The Functional Block of the CCG – the VHDL Post-Processing Tool – generates a Functional Configuration Controller that allows the designer to perform the functional simulation of dynamic designs using static simulators, with little effort. In this model, the effect of dynamic reconfiguration is represented by surrounding all dynamic modules’ inputs and outputs with isolation switches that are controlled by Schedule Control Modules, *SCMs*. The control of the isolation switches is distributed among the *SCMs*, which collectively represent a Functional Configuration Controller, *FCC*.

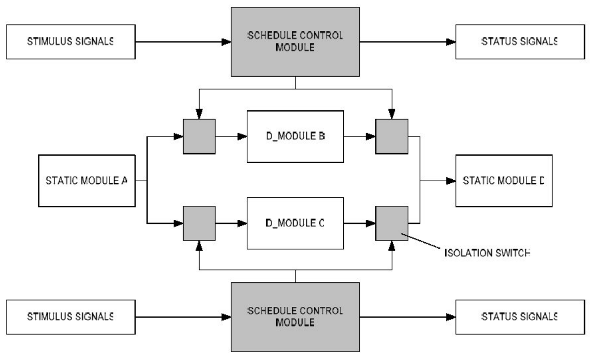

Fig 7 Example design including the modifications done by the VHDL Post-Processing Tool

The simulation strategy followed by the tool is inspired in the Dynamic Circuit Switch technique introduced in [0], [[2]], [[3]]. To be able to perform the simulation depicted in the Fig 7, a mechanism has been implemented which adds switches to the dynamic modules. There are two types of generated switches: 1-bit and *n*-bit, where *n* is the generic representing the length of the signal. Fig 8 presents 1-bit switch and TABLE 1 depicts how it is controlled – A and B ports may be both inputs or outputs (one is input, the other is output), as switch is bidirectional; for ON state a resolved function from the IEEE 1164 VHDL package body is used. The implementation of these isolation switches is done by means of VHDL descriptions.

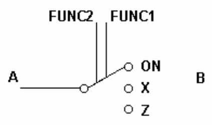

Fig 8 1-bit switch structure. For n-bit switch, there are also FUNC1 and FUNC2 input signals which controls all bits together

TABLE 1 Explanation of the switch controlling

State	FUNC2	FUNC1	A port	B port
ON	0	1	Resolved	Resolved
X	1	1	X	X
Z	1	0	Z	Z
Z	0	0	Z	Z

The switch is able to transmit signal from its input to output – it corresponds with the dynamic module turned on, as well as it is able to stop the signal – it corresponds with the dynamic module turned off. Moreover, there is a possibility to model reconfiguration by setting an unknown value to a switch inputs/outputs (and inputs or outputs of a dynamic module at the same time). To manage the switch two control signals are provided: FUNC1 and FUNC2. These signals have to be decoded and this determines the state of the switch, as it is presented in the TABLE 1.

The VHDL Post-Processing Tool needs as its inputs VHDL files containing dynamic modules description; cut entities description together with static part of the design. Moreover, it requires a DCF file to be able to understand the dynamic behaviour of the design – all of these files are produced by the Manual Partitioning Tool. As the result, the application generates a VHDL description of the FCC, switches used to isolate the dynamic modules, these modules with added switches as components, top entity putting everything together and model simulation script file (*.do). Moreover, the tool generates a set of internal configuration signals, which are useful to assert in which state the generated controller actually is.

Technology Dependent Block

This block is in charge of the generation of the Reconf Interface – an interface between the static part of the design interface (Application Interface), FPGA device interface (Reconfiguration Interface) and Isolation Switches (described previously, used for verification purposes in this case). Fig 9 shows the Configuration Controller Generator design environment view. Both static part and dynamic modules are the inputs for the CCG, while Reconf Interface is its output. Isolation switches are used only for verification purposes.

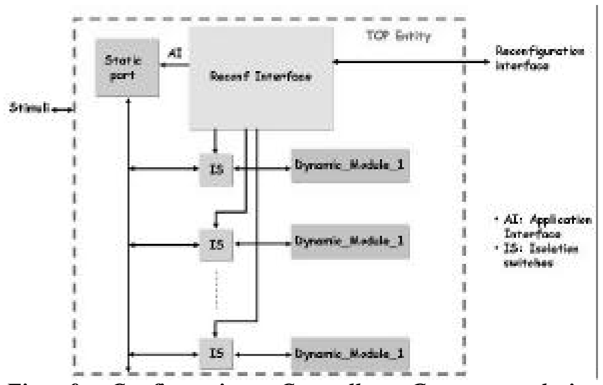

Fig 9 Configuration Controller Generator design environment

One may distinguish the following entities as a part of the Reconf Interface, as it is depicted in the Fig 10:

Event Detector. This block reflects the conditions for loading or unloading the dynamic modules as specified in the *DCF* file relative to the design.

Sequential Scheduler. This entity is responsible for sequentially loading dynamic bitstreams accordingly to the events detected by the event detector.

Physical Interface. This entity is responsible for managing internal or external reconfiguration ports of the D_FPGA and internal/external bitstream memory. The part of the interface that is communicating with the FPGA is device dependent.

Physical Configuration Controller. This entity is responsible for sequentially requesting all the bitstream data of the bitstream associated to the bitstream ID requested by the *Sequential Scheduler*. It is also responsible for reading the start and end pointers of the bitstream to load. This is the main configuration controller part that is to be implemented in the hardware.

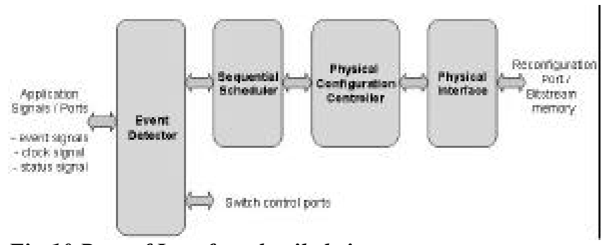

Fig 10 Reconf Interface detailed view

The Physical Configuration Controller has two running modes: an initialization mode and a run-time mode. During the first one, the *PCC* has to load on the device the static part plus some dynamic modules. In the run-time mode, the *PCC* has to monitor a set of signals and detect the triggering events for the loading and unloading of dynamic modules. Once these events are detected the configuration controller proceeds to load the dynamic modules and to manage the context data and interface data management.

The Technology Dependent Block has the same inputs as the Functional one, since they are implemented in the same tool. The application generates a *PCC* description in C code (for an external configuration controller, in this case the controller is represented by a set of instructions to be executed on an external micro-controller) and VHDL description of the Reconf Interface, described above.

DESIGN FLOW EXAMPLE

Fig 11 presents the example of the input VHDL design that may be loaded to the Manual Partitioning Tool, since it is the first tool in the presented design methodology. The "P*x*" circles represent VHDL processes with their inputs and outputs, where *x* is a natural number. As it may be observed there, static description of the project has three input signals with one output.

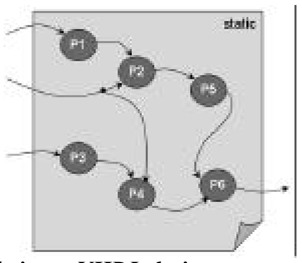

Fig 11 Example input VHDL design

Fig 12 depicts the situation after Manual Partitioning Tool modifications. There are two dynamic modules set (d_module_1 and d_module_2), both of them with two processes. Moreover, a cut entity has been created, since the output of the dynamic module 1 is an input for the dynamic module 2 (data management needs to be undertaken). The initial static design interface depicted in the Fig 11 has been extended to the interface between static and dynamic parts of the design. The output VHDL files of the Manual Partitioning Tool should be next loaded to the Configuration Controller Generator.

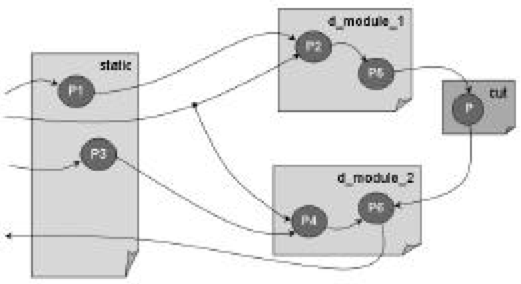

Fig 12 Modification done by the Manual Partitioning Tool

Fig 13 depicts only one possible modification done by the Configuration Controller Generator – the design preparation for the functional simulation. In this case the Functional Configuration Controller (FCC) is responsible both for switch (S) and cut entities (CUT) controlling according to the event signals. Event signals may be the inputs or outputs of the static part of the project, or inputs of the top level file, since all of the depicted files will be merged at the end to the top level file.

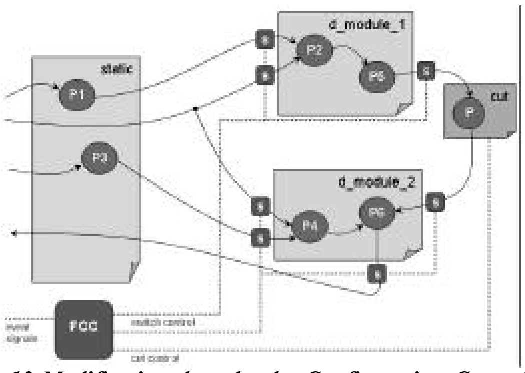

Fig 13 Modification done by the Configuration Controller Generator, Functional Block

CONCLUSION

The main goal of this work was a presentation of the Front-End Tools for dynamic reconfigurable FPGA devices. The developed environment forms the basic interface between the designer and the device. It allows the user to decide in details about the dynamic reconfiguration characteristic. At the beginning, it is possible to use both automatic and manual partitioning algorithm in the system scheduler. The user may simulate the dynamic functionality of the system using the currently available static simulators. It is possible to generate the configuration controller that is going to be implemented in the hardware or used for functional simulation. At the end, all the project's files may be merged together in one top-level file.

The current efforts are concentrated on the integration of the presented tools into one application, *IFDYR – Integrated Front-end for DYnamic Reconfiguration*. It would ease up the design flow from the user's point of view. Moreover, the application could be fully parameterized, which means that the designer would have fully influence on project details, such as generated signals naming, to enumerate only the basic example.

THE AUTHORS

MSc. Kamil Kedzierski, Prof. Juan Manuel Moreno-Arostegui and Prof. Joan Cabestany are with the Advanced Hardware Architecture Group, Department of Electronic Engineering, Technical University of Catalonia, c/Jordi Girona 1-3, Barcelona, SPAIN

REFERENCES

[1] J. Stockwood, P. Lysaght, "A Simulation Tool for Dynamically Reconfigurable Field Programmable Gate Arrays", IEEE Transactions on VLSI Systems, 4-3 (1996), 381 -390

[2] G. McGregor, P. Lysaght, "Extending Dynamic Circuit Switching to Meet the Challenges of New FPGA Architectures", Field Programmable Logic and Applications, Springer-Verlag (1997)

[3] I. Robertson, J. Irvine, P. Lysaght, D. Robinson, "Improved Functional Simulation of Dynamically Reconfigurable Logic", Field Programmable Logic and Applications Lecture Notes in Computer Science Vol. 2438, Springer-Verlag (2002), 152-161

[4] K. Kedzierski, J.M. Moreno, J. Cabestany, "Constraints Editor, Technical Requirement Specification", UPC (2002), available through www.reconf.org

[5] K. Kedzierski, J.M. Moreno, J. Cabestany, "VHDL Post-Processing Tool. Technical Requirement Specification", UPC (2002), available through www.reconf.org

[6] K. Kedzierski, J.M. Moreno, J. Cabestany, "Configuration Controller Generator. Technical Requirement Specification", UPC (2002), available through www.reconf.org

[7] R. Kielbik, J.M. Moreno, A. Napieralski, T. Szymanski, "High-Level Partitioning for Dynamically Reconfigurable Logic", Proceedings of the 7th International Conference Mixed Design of Integrated Circuits and Systems (MIXDES'2000), 171-174, Gdynia, Poland, June 15-17, 2000

[8] R. Kielbik, J.M. Moreno, A. Napieralski, G. Jablonski, T. Szymanski, "High-Level Partitioning of Digital Systems Based on Dynamically Reconfigurable Devices", Field-Programmable Logic and Applications. Reconfigurable Computing is Going Mainstream, M. Glesner, P. Zipf, M. Renovell (eds.), pp. 271-280, Springer-Verlag, Montpellier, France, September 2-4, 2002