TOWARDS SYNTHETIC BENCHMARKS GENERATOR FOR CAD TOOL EVALUATION

Mariem Turki
Habib Mehrez
LIP6, Paris 6
Email: mariem.turki@lip6.fr

Zied Marrakchi
FlexRAS Technologies
Paris, france
Email: zied.marrakchi@flexras.com

Mohamed Abid
CESlab
Sfax, Tunisia
Email: mohamed.abid@ceslab.org

***Abstract*—In the process of designing prototyping CAD tools for system on chip, industrial need usually huge benchmark designs to test and evaluate their tools.**
However, observing the limits of existing and available benchmarks, it is necessary to develop and provide large suites of synthetic benchmarks with more specific characteristics.
In this paper, we present a benchmark generator providing various sets of architectures. The generation includes also the creation of the executable code of the application which will be executed in the hardware architecture.
In the experimental results, we show that our generator is able to generate various sets of synthetic benchmarks with several millions of gates. Those sets includes hierarchical, heterogeneous and asynchronous circuits.

I. INTRODUCTION

With the global trend towards digital systems, designers studies are focused to manage the design complexity in the available design time that is ever decreasing due to tightening time-to-market constraints.
One interesting feature to decrease the time to market is to validate the increasingly large designs erlier before reaching the manufacturing phase. This feacture is called multi-FPGA hardware prototyping.
However, the big challenge of prototyping CAD vendors is to find specific benchmarks to validate and test their tools and algorithms. Indeed, the requested benchmarks have to be various and big enough in order to be tested in a multi-FPGA board. The variety is assured by generating designs with a mix of homogeneity and heterogeneity. One other constraint of the requested benchmarks is the testability. In other words, user should always have controlability and observability on the circuit under test.
The first initiative to generate such circuits was made by CBL[1] [1], [2] and MCNC[2][3]. However, those benchmarks are not big enough to target the current challenges of prototyping CAD tools which require netlists with up to several millions of gates. Indeed, the biggest circuits given by CBL is s38584 netlist and it contains only 2904 configurable logic blocks (CLBs) [4].
More recently, researchers developed a benchmark generation program GNL[3] [5] which generates netlists with more realistic behavior. This program is based on Rent's rule [6] to control the interconnection complexity. Indeed, the user defines the number of gates, flip flop, the number of primary inputs, outputs and also the rent exponent. GNL uses a bottom-up approach, so it starts by establishing connections between a set of gates which allows to create a number of clusters. The clusters themselves are recursively paired further with other clusters until all clusters are combined to one circuit. When doing the connections, the program ensures the respect of the Rent exponent at every level. The problem of this method, is that since the assignment of the functionality to the gates is still done at random, the generated circuits are highly redundant.
In 2005, the IWLS benchmarks suite was published by International Workshop on Logic and Synthesis (IWLS)[7]. It contains diverse circuit designs derived from past conference benchmarks, open source community of hardware designers, and industry to represent a variety of applications. Although these circuits are relatively big, but they do not offer any solutions for testability during the implementation in the multi-FPGA board.
In this paper, we seek to generate more realistic circuits. Whereas previous efforts for benchmark generation mainly focused on graph-based properties of circuits, we wish to generate bigger architectures based on open library IP. Our work targets circuits with synchronous and asynchronous clocking.
To reach this goal, we modified some existing tools to make the generation more fast and especially automated.
In section 2 we give a brief description about the hardware prototyping steps. Section 3 describes the proposed benchmark generation framework. It includes the hardware and the software toolchains. Section 4 states the proposed architectures details. Asynchronous features included in the generated benchmarks are covered in section 5. Finally, experiments and results are presented in section 6.

II. HARDWARE PROTOTYPING

Many design and verification teams are increasingly using multi-FPGA prototyping to meet ever decreasing time-to-market constraints. Prototyping includes two steps:

[1]Collaborative Benchmarking Laboratory, North Carolina State University,Raleigh, NC.
[2]Microelectronics Center of North Carolina
[3]gnl is the acronym for Generate NetList

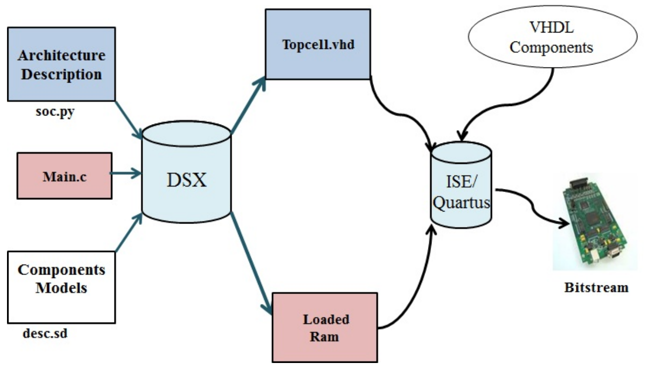

Fig. 1. DSX toolchain

partitionnig and routing. The partitioning algorithms search for the best partition with the lowest inter- FPGA connections abd highest system performance. The output of this task is a new design hierarchy which highlights the different partitions for each FPGA.
The routing algorithm routes inter-FPGA and I/O signals through board traces or FPGA's with the objective of minimizing signal delays. The Outputs of this task includes individual projects for each FPGA containing each the multiplexing IP for signals in the FPGA interface.

III. Benchmark generator framework

The Design Space eXploration (DSX) tool[8] allows the co-design of hardware platforms based multiprocessor architectures on chip (MP-SOC). Initially, DSX uses component modules provided by the library SoCLib[9]. The modules are written in systemC and yet the generated description file of the platform is written in systemC.
In a second step, we modified DSX to generate, in addition to the SystemC simulator caba, a synthesizable VHDL platform and implementable on FPGA board. The initial toolchain (systemC) and the working environment have been modified in order to facilitate the generation of the systhesizable VHDL netlists. In the Fig 1, we present the proposed toolchain.
To generate specific architectures, the user have to set the platform characteristics in the input files.

A. Input files

Three files are needed to generate the architecture. The first one is the platform description file which is written in python language. This file contains the instantiation of each component in the architecture with the specific parameters.
For example, when instantiating the simple ring bus which connects all components, user need to specify the number of initiators, the number of targets, the size of data etc.. This file contains also the connections between all components.
This step is made using simple commands thanks to metadata file (.sd) which contains a detailed description of the interface of each component. All the ports of each module are enumerated with related details such as the port type, size, name etc..
For example, let's cite the VCI protocol. The user is not obliged to connect all the signals of this protocol. In the metadata file, the VCI port isdeclared as a composed port which contains many related signals like shown in Fig 2. So, when the user connects the bus to any other component, he have to mention only the connection between the composed ports of those components, and the related signals will be connected automatically. In other words, the connection between the component and the ring is done by the following simple line:
VCI_ring.port.vci // component.port.vci
The last input file is the software application. The applications should be written as multiple communicating tasks (threads). This script can easily distribute the application on the platform. The goal is to get applications with a variable number of tasks running on a platform having itself a variable number of processor / coprocessor. The application must allow accurate testalibility. Indeed the user is able to observe results using leds on the board or using messages on the screen sent by the uart component. In the other side, the processors can send patterns for the different coprocessors and test the results with references saved in the memory.

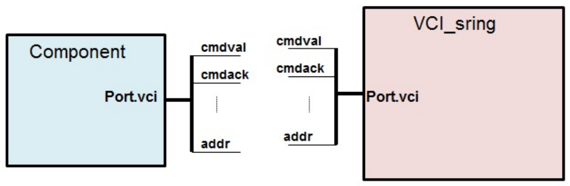

Fig. 2. Example of composed port

B. Output files

After preparing all the input files needed to create the architecture, DSX generates the related output files.
The first output is the VHDL Ram file. We use a software toolchain called by DSX. This toolchain is represented in Fig 3.
The software application is compiled with a cross compiler

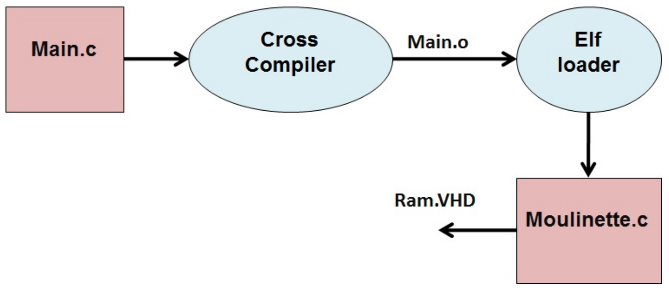

Fig. 3. Software toolchain

related to the processor in the architecture. The binary file will next pass through an elfloader to extract the content of every segment. The final step is to generate a vhdl file which instantiates the considered ram-block with the executed code. The VHDL Ram file, the topcell file and the VHDL components are the inputs of the corresponding vendor environment (ISE/quartus) to generate the bitstream which will be loaded into the FPGA.

IV. Benchmark architectures

A. monocluster architecture

The proposed benchmarks are multiprocessors based architecture and contains also many coprocessors. The goal of generating such architectures is to obtain asymmetric and decentralized benchmarks. Those architectures represent a mix of homogeneity(multiprocessors) and heterogeneity(multi-coprocessors). An example of this architecture is represented in Fig 4.

This architecture contains a set of components which communicate via a VCI protocol. All those components are connected to a ring bus which is parameterizable, so the user can easily set the number of targets, initiators etc..

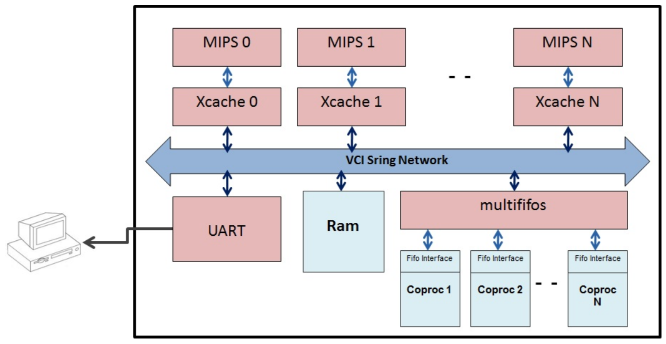

Fig. 4. Example of generated architecture

The example in the Fig 4 contains N processors and 3 targets: Ram, uart and a parameterizable multi-fifos component. The multififos acts as a bridge between the coprocessors, each with a fifo interface, and the ring network. The user can use a large set of coprocessors in order to have the biggest design. To add more suppleness to our design, and to make it more realistic, we choose to integrate an embedded FPGA in the multicoprocessor architecture. Indeed, more recent SOC contains some field programmable cells in order to reuse a portion of the chip and to introduce new features in the design even after its fabrication.

In addition, FPGA vendors and new IP developers are now offering hard embedded FPGA core that can be added into a SOC design[10], [11].

The embedded FPGA which we included in our architecture has been developed into our laboratory[12].

The Fig 5 shows the connexion interface between the e-FPGA and the multi-fifos component.

To communicate with the e-FPGA, we need only one read fifo and one write fifo. The first one is used to configure our e-FPGA and also to select the inputs. The write fifo is used to recover the resulting outputs.

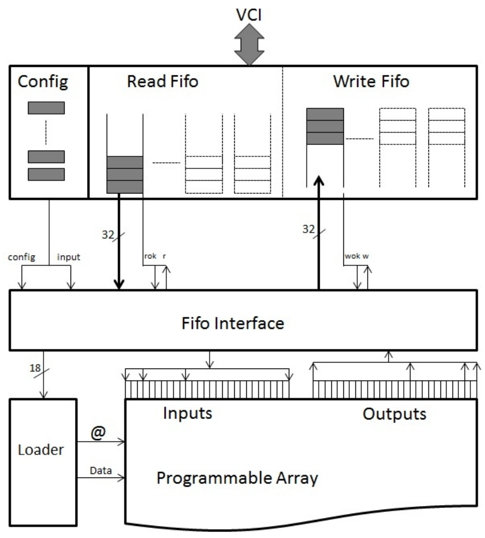

Fig. 5. e-FPGA interface

B. multicluster architecture

Experimentally, when we increase infinitely the number of processors, we are faced to the bandwidth limitation. For this reason, we choose to pack processors into clusters and create a two-level interconnect. The global (inter-cluster) interconnect will use the DSPIN [17] Network on Chip that supports the VCI standard. the local interconnect uses a simple ring network with gateways to communicate with the global network.

The DSPIN network on chip has a 2D mesh topology and provides a truly scalable bandwidth. Each node in this mesh represents a router and its corresponding cluster (called sub-system). The router has five modules. Four of them are placed on the north, south, east, and west side of the subsystem in order to route the packets between the clusters in the horizontal and vertical sides. Finally, a local module which communicates to the local subsystem through the Network Interface Controller (NIC).

V. Asynchronous clock

Most of recent system on chip are asynchronous designs. So, as the CAD tools are more and more supporting the asynchronous methodologies, we propose here to integrate the asynchronous factor in our benchmarks.

The idea is to insert a bi-synchronous fifo between the VCI local bus and the VCI uart component.

Actually, we decided to keep the VCI interface of the uart component, and we tried to transfer all the control signals of the VCI protocol trough the bi-synchronous fifo.

The Fig 6 shows the connection between the the network and

TABLE I
PROTOTYPING RESULTS OF SOME GENERATED BENCHMARKS

Benchmark	LUTs	RAMLUTs	DSP	RAM	REG	NB FPGA	MUX ratio	Freq(MHz)
CPU_20	143217	6192	2	21	66937	1	1	80
CPU_30	213524	9272	2	21	99588	3	9	27,78
CPU_50	353697	15432	2	21	164587	4	12	20,83
CPU_125	879897	38532	2	21	408712	5	17	14,70

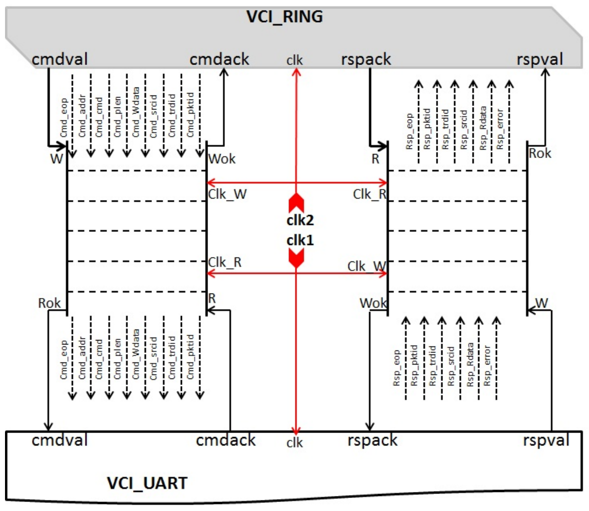

Fig. 6. Integration of an asynchronous fifo

the fifo from one side, and between the fifo and the uart from the other side.

VI. EXPERIMENTS AND RESULTS

We evaluated several synthesis tools such as Synopsys's Synplify Pro[13] and Xilinx's XST[14] for the targeted boards. Since our benchmarks are relatively big, the primary importance was to select a tool which could synthesize the design in an acceptable runtime.
However, XST was not able to synthesize our generated designs which exceed some hundred of thousands of LUTs. The synthesis process always ends with an out of memory error.
So we select synplify pro as a synthesis tool since it has the compile point feature which allows a fast synthesis runtime. The idea is to divide the design into different parts or points that can be processed independently. We select the components which forms the biggest part in the design. These are the first to be synthesized starting with the block at the lowest level of hierarchy in the design. After all the compile points are synthesized, the software synthesizes the design from the top down, using the model information for each compile point.
Our benchmarks are used by Flexras technology's prototyping tools[15]. The results of implementation of some benchmarks in a multi-FPGA board are presented in table I. The board used to prototype our designs contains six virtex-6 FPGA(xc6vsx475tff1759)[16]. TableI shows the details related to each benchmark such as lut size, register numbers etc..After partitioning and routing the designs into the multi-FPGA board, information about the frequency and the number of used FPGAs are given by the prototyping tool.

VII. CONCLUSION

In this paper, we presented the framework of a synthetic benchmark generator. Our generator is able to design various set of circuits in a very small time using an existing IP library. the generation includes the hardware and the software parts of the circuit.
The resulting benchmarks are suitable for multi-FPGA prototyping CAD tool evaluation since they include requested properties like hierarchy, asymmetry, big size and asynchronous clocking.

REFERENCES

[1] Computer aided design benchmarking laboratory, http://www.cbl.ncsu.edu/benchmarks/.
[2] C. J. Alpert, "The ispd circuit benchmark suite", in Proc. ACM/SIGDA Intl. Symp. on Physical Design, 1998, pp. 85- 90.
[3] "layout synthesis benchmark set", microelectronics center of north carolina, research triangle park, NC, May 2006.
[4] R. Kuznar, F. Brglez, and K. Kozminski, "Cost minimization of partitions into multiple devices", in In Proc. 30th ACM/IEEE Design Automation Conf, June 1993, pp. 315- 320.
[5] D. Stroobandt, P. Verplaetse, and J. van Campenhout, "Generating synthetic benchmark circuits for evaluating cad tools", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 19, pp. 10111022, Sept. 2000.
[6] B. S. Landman and R. L. Russo, "On a pin versus block relationship for partitions of logic graphs", IEEE Trans. on Comput, vol. C.20, pp. 14691479, 1971.
[7] C. Albrecht, "Iwls benchmarks", www.iwls.org/iwls2005, 2005.
[8] N. Pouillon and A. Greiner, URL=https://wwwasim. lip6.fr/trac/dsx/, 2006-2008.
[9] "Soclib project: Platform for modeling and simulation of integrated systems on chip", http://www.soclib.fr/.
[10] M. Inc, "Menta efpga core-ii data sheet brief", http://www.menta.fr/down/DatasheetBrief-eFPGA-core- II.pdf, Feb. 2009.
[11] "M2000 intros largest 90nm efpga, design and reuse", http://www.design-reuse.com/news/9614/m2000- introslargest-90nm-efpga.html, Feb. 2005.
[12] http://www-soc.lip6.fr/recherche/cian/.
[13] Synopsys FPGA Synthesis User Guide, 2011.
[14] Xilinx. xst. www.xilinx.com/products/design tools/logic design/ synthesis/xst.htm
[15] www.flexras.com.
[16] www.dinigroup.com/new/dnv6f6pcie.php.
[17] I. Miro-Panades and A. Greiner and A. Sheibanyrad, "A Low Cost Network-on-Chip with Guaranteed Service Well Suited to the GALS Approach", 1st Int. Conf. on Nano-Networks and Workshops, Sep 2006.