Multi-FPGA Prototyping Environment: Large Benchmark Generation and Signals Routing

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 multi-FPGA prototyping systems for circuit verification, serialized time-multiplexed I/O technique is used because of the limited number of I/O pins of an FPGA. The verification time depends on the number of inter-FPGA signals to share the same physical wire and be time-multiplexed. In this paper, we propose an adaptation of Pathfinder routing algorithm that minimizes the verification time of multi-FPGA systems by reducing the multiplexing ratio per physical wire. To run real experiments, we propose a large benchmark generation environment and we show that the verification system clock frequency is improved by 17% on average compared with conventional methods.**

I. Introduction

Circuit verification is one of the essential processes of VLSI design. In circuit verification, it is often requested to use very large test benches. In such cases, software RTL simulations are too slow to adopt, and FPGA prototyping systems are used. The verification time tends to increase according to the increase in size and complexity of VLSIs. The desire to shorten verification time is becoming stronger.

Although the device capacity of FPGAs is becoming very large, circuits to be designed are often larger than the leading edge FPGAs. Therefore, a prototyping system for such circuits consists of multiple FPGAs. A large circuit is partitioned into several sub-circuits each of which is implemented into an FPGA. Each sub-circuit has to communicate with other sub-circuits through I/O pins of the FPGA. The number of I/O pins of an FPGA is becoming larger. However, the number of inter-FPGA signals from a sub-circuit is usually larger than the number of I/O pins of an FPGA even if a circuit is partitioned so that the number of inter-FPGA signals is minimized. Moreover, there exists a signal path that is cut several times by partitioning even if its minimization is pursued. Our goal is to propose software CAD tools to improve performances of the design under test implemented on multiple FPGA board. Thus, we need various and large benchmarks in order to validate proposed tools. To cover a large spectrum of cases, benchmark designs must be realistic, large enough, have heterogeneous architectures and be testable. In fact, user should always have controllability and observability on the circuit under test.

The first initiative to generate such circuits was made by CBL [1], [2] and MCNC [3]. However, these benchmarks are not large 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 tool GNL [5] which generates netlists with more realistic behavior. This tool 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 in a random way, the generated circuits are highly redundant. In addition such random generated circuits are not testable since the generator does not provide input and output test patterns.

Due to this lack of complex and realistic benches, we propose to generate such design circuits. The proposed benchmark generation framework is described in section 2. It includes the hardware design and the application software flows. Section 3 and 4 describe the proposed design architectures details and features. In section 5 and 6 we propose a multi-FPGA prototyping flow and we introduce an innovative technique to route inter-FPGA signals. Finally, experiments of the routing technique are presented in section 7. Results are obtained based on the generated benchmark circuits.

II. Benchmark generator framework

The Design Space eXploration (DSX) tool [7] allows the co-design of multi-procrssors based hardware architectures (MP-SOC). DSX uses component modules provided by the SoCLib library [8]. The modules are written originally in systemC and yet the generated description file of the platform is written in systemC.

We extended DSX to generate, in addition to the SystemC simulator caba, a synthesizable VHDL platform that can target FPGA implementation. The initial systemC flow and the working environment have been modified in order to facilitate the generation of the synthesizable VHDL netlists.

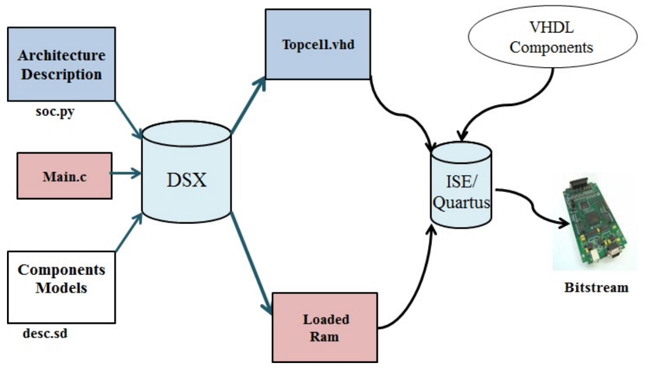

Fig. 1. DSX framework

In the Fig 1, we present the proposed generation flow.
To generate specific architectures, user has 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, to instantiate the simple ring bus which connects all components, designer needs to specify the number of initiators (masters), the number of targets (slaves), the data width etc.. This file contains also the description of 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, sub-ports etc..
For example, in the case of VCI (Virtual Component Interface) protocol. The user is not obliged to connect all the detailed signals of this protocol. In the metadata file, the VCI port is declared as a composed port which contains many related signals as shown in Fig 2. So, when the user connects the bus to any other component, he has to mention only the connection between the composed ports of those components, and the related signals will be connected automatically. For example, the connection between the component and the ring is done by the following expression:
VCI_ring.port.vci // component.port.vci
The third input file describes the software application graph. The applications should be written as multiple communicating tasks (threads). In this way, we 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 check functional results using LEDs on the board or using messages on the screen via serial communication between the prototyping board and PC.

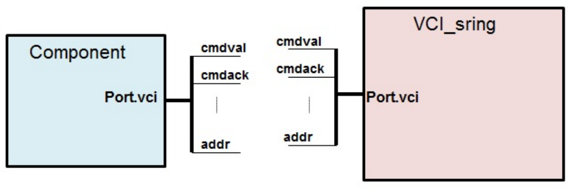

Fig. 2. Example of composed port

B. *Output files*

After preparing all the input files required to create the architecture, DSX generates the resulting output files.
The first output is the VHDL Ram file. We use a Ram loader called by DSX. This tool chain is presented in Fig 3.
The software application is compiled with a cross compiler

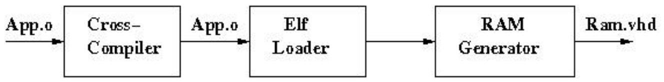

Fig. 3. Generation of loaded Ram module

related to the processor in the architecture. The binary file passes through an ELF loader to extract the content of each memory segment. The final step is to generate a vhdl file which instantiates the corresponding ram-block with the executed code targeting a specific FPGA memory organization.
The VHDL Ram file, the topcell file and the VHDL components are the inputs of the corresponding FPGA environment (ISE/quartus) [12] in order to generate the configuration bitstream.

III. Benchmark architectures

A. *mono-cluster architecture*

The proposed benchmarks are multiprocessors based architecture and contain also several coprocessors. These architectures represent a mix of homogeneity (multiprocessors) and heterogeneity (multi-coprocessors). An example of such architecture is represented in figure 4.
The generated architecture contains a set of components

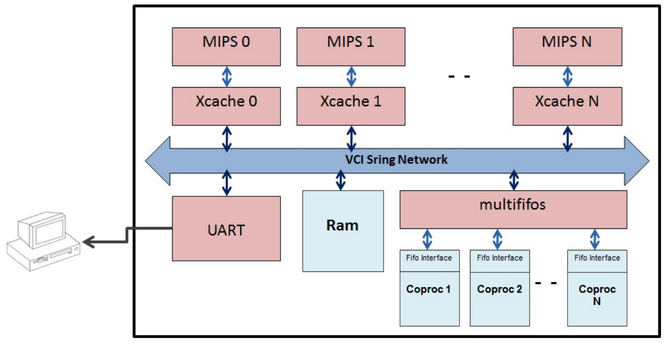

Fig. 4. Example of multi-processors / multi-coprocessors architecture

which communicate via a VCI (Virtual Component Interface) protocol. All these components are connected to a generic ring bus, so the user can easily set the number of targets and initiators. The example in the Fig 4 contains N processors and 3 targets: Ram, UART and a generic multi-fifos component. The multi-fifos acts as a bridge between the coprocessors, each with a fifo interface, and the ring network. The designer can use a large set of coprocessors in order to have the biggest design.
To add more complexity to the generated design we integrated an embedded FPGA in the multicoprocessor architecture. Indeed, 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 [9]. The embedded FPGA which we included in our architecture has been developed by [10]. Figure 5 shows the connection interface between the eFPGA and the multi-fifos component.
To communicate with the eFPGA, we use one read fifo and

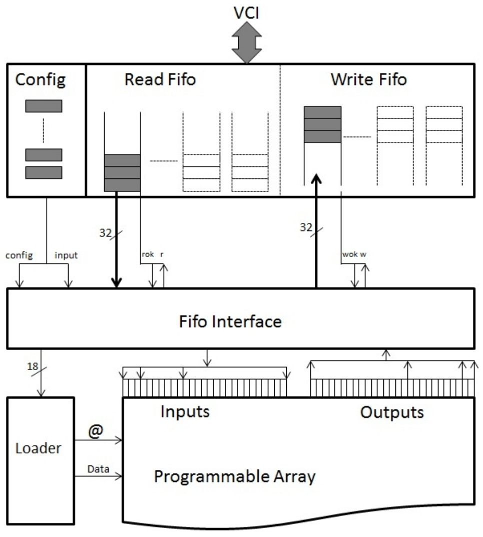

Fig. 5. Embedded FPGA (eFPGA) interface

one write fifo. The first one is used to configure the eFPGA and send the input patterns. The write fifo is used to send back the resulting outputs via the bus.

B. multi-clusters architecture

When we increase infinitely the number of processors, we are faced to the bus bandwidth limitation. For this reason, we choose to pack processors into clusters and create a two-level interconnect architecture. The global (inter-cluster) interconnect uses the DSPIN [15] Network on Chip that fits the VCI standard. The local interconnect uses a simple ring network communicating 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

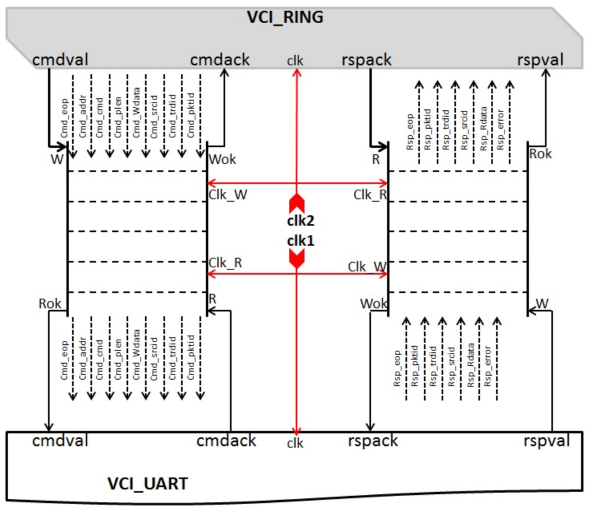

Fig. 6. Integration of an asynchronous fifo

represents a router and its corresponding cluster (called subsystem). 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.

IV. Multi-clock domains

Most of recent systems on chip use multiple clocks domains. To be closer to reality we added this property to the generated designs. The idea is to insert a bi-synchronous fifo between the VCI local bus and the VCI external interface components (UART for example). Actually, we decided to keep the VCI interface of the UART component, and we transfer all the control signals of the VCI protocol trough the bi-synchronous fifo.
Figure 6 shows the connection protocol between the network and the fifo from one side, and between the fifo and the UART from the other side.

V. Prototyping Flow

The input, a netlist of the logic design (generated architecture) to be prototyped, is transformed into a multi-FPGA configuration bitstream to be downloaded onto the prototyping board. As shown in figure 7, the technology libraries, target FPGA characteristics, and FPGA interconnect topology are required to make the correct implementation.

A. Logic Synthesis

The input design netlist is mapped to a target library of FPGA primitives. In our case we use commercial and public domain tools for mapping [11].

B. Partitioning

After mapping the netlist to the target technology, it is divided into partitions, each of which can fit into a single target FPGA. This partitioner performs K-way partitioning with multi-objectives function allowing to find the best tradeoff

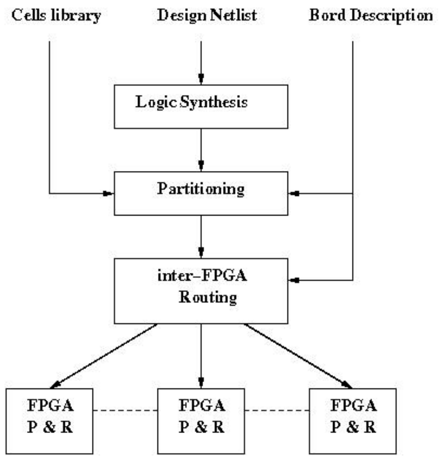

Fig. 7. Multi-FPGAs Prototyping Flow

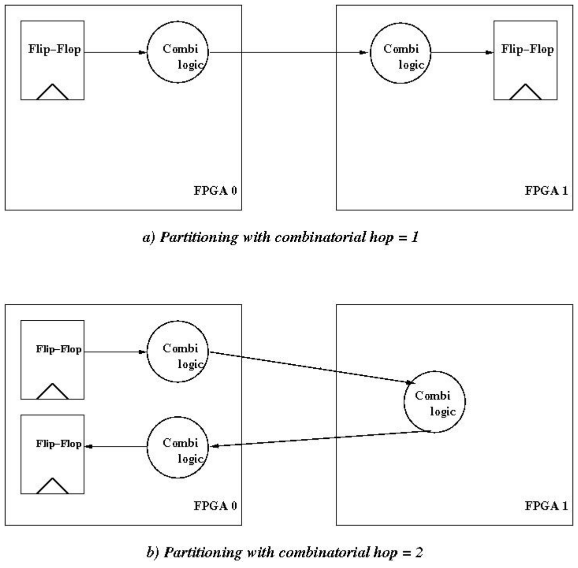

Fig. 8. Combinatorial hops example

between:
- Minimizing the maximum slots number: the ratio between communication signals and available physical between each FPGA pairs
- Reducing the maximum combinatorial hops: number of times a combinatorial path exists an FPGA (see figure 8)
- Reducing the maximum routing hops: number of FPGA to cross to route a signal between 2 FPGAs (see figure 9).

C. Routing and Multiplexing

In order to relax the I/O pins constraint, time multiplexed I/O (TM I/O) technique, such as in [16], is used in multi-FPGA prototyping systems. When TM I/O technique is employed, an I/O pin is used as a TM I/O, which is shared by multiple signals by time-division. Multiple inter-FPGA signals are transmitted by a TM I/O in one system clock period. An inter-FPGA signal transmission is far slower than an intra-FPGA signal transmission, and the required time to complete inter-FPGA signal transmissions by a TM I/O is far larger than that by a normal I/O. Therefore, the system clock period is mainly bounded by inter-FPGA signal transmissions (TM I/Os). Thus, we propose as shown in figure 10 an iterative routing technique to reduce multiplexing ratio. In this loop we decrease the multiplexing ratio MR , we create group of MR branches, called meta-branches, having the same source and destination and then we try to route them. When the multiplexing ratio is decreased the number of meta-branches to route is increased and the router has to make more effort to succeed. Consequently, the routing algorithm has to be very efficient to deal with the increasing congestion.

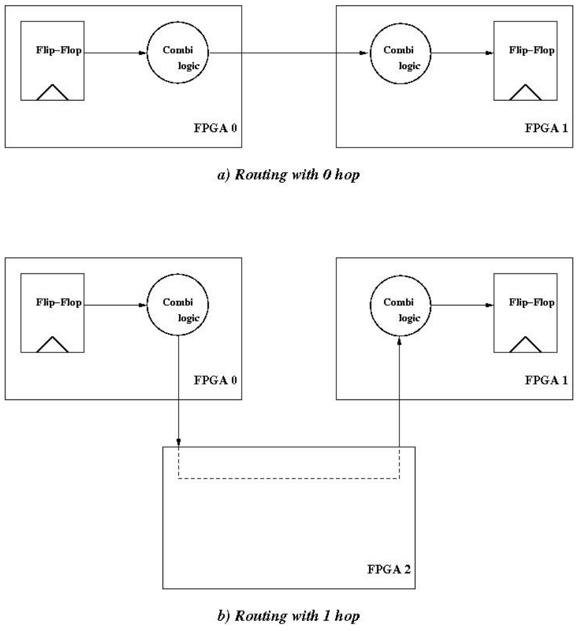

Fig. 9. Routing hops example

D. FPGA Place & Route

Once routing is achieved, there is one netlist for each FPGA. Each netlist must be processed with FPGA specific automated place-and-route (P&R) software to generate configuration bit-streams.

VI. Routing Strategy Description

The goal of the routing algorithm is to find a shortest available path, in terms of FPGAs, between the source and destination FPGA of a set of inter-partition branches. A signal of fan-out N is transformed into N branches (bi-points). Branches are grouped into MR meta-branches where MR is the target multiplexing ratio. Then, meta-branches graph is routed using available physical wires.

A. PathFinder adaptation

Common inter-FPGA routing algorithms are based on obstacle avoidance techniques. For example, the router proposed in [16] proceeds as follows. Before the beginning of routing a reservation matrix is initialized to the number of physical

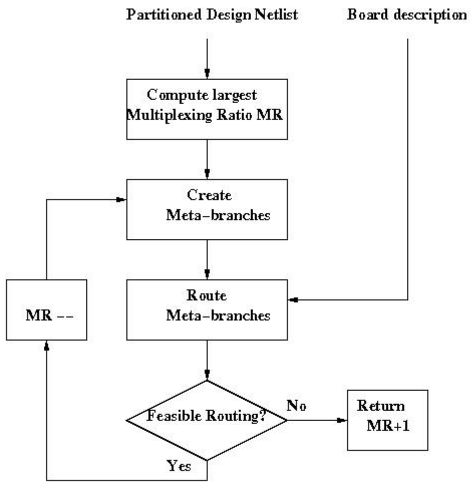

Fig. 10. Iterative Routing to minimize Multiplexing-Ratio

connections between FPGAs i and j in the board. Router applies Dijkstra's shortest path algorithm [17] to determine the shortest path between the source and destination FPGAs. If the shortest path exists, then the reservation matrix is updated by subtracting 1 from each element along that path and router returns with this path; else, router returns unsuccessfully. This technique has the disadvantages to be dependent on the order in which branches are routed. As we stated previously, the efficiency of reducing the multiplexing ratio depends on the quality of the router. Thus the routing algorithm we proposed is an adaptation of "PathFinder" [18]. PathFinder were widely used to achieve intra-FPGA signals routing [19]. It uses an iterative, negotiation-based approach to successfully route all meta-branches in a netlist. During the first routing iteration, meta-branches are freely routed without paying attention to resource sharing. At the end of iteration, resources can be congested because multiple meta-branches use them. During subsequent iterations, the cost of using a resource is increased, taking into account the number of meta-branches that share the resource, and the history of congestion on that resource. Thus, nets are made to negotiate for routing resources.

B. Routing graph

To apply PathFinder, we model the available board routing resources as a directed graph abstraction $G(V,E)$. As illustrated in figure 11, the set of vertices V represents the pins (pads) of FPGAs. An edge between two vertices represents a physical wire between 2 FPGA pins. Since inter-FPGA connections are bi-directional each connection is presented by two opposite directed edges. Since FPGA pins are equivalent we add for each FPGA a virtual node (nodes F0, F1 and F2 in figure 11). Each virtual FPGA node has pair of edges to all other FPGA pins node. A routing path starts from a source virtual node and ends at a destination virtual node.

VII. Experiments and results

To evaluate the proposed routing technique we implemented several generated designs on an industrial

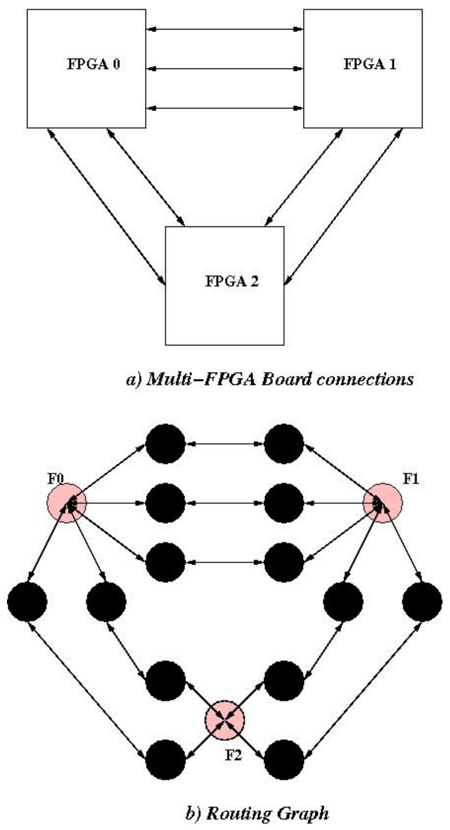

Fig. 11. Routing Graph description

Dini board DNV6F6PCIe [14] which contains 6 Virtex-6 (XC6VSX475TFF1759). In table I, we present the characteristics of the board in terms of logic resources capacity. We show also the amount of resources required by each synthesized design. Designs names correspond to the number of processors they contain. To add heterogeneity we include various coprocessors in each architecture. Each design is partitioned, routed and implemented on the board. In table I, we show the partitioning result of each design in terms of logic resources occupancy, number of used FPGA, Cut (number of signals crossing FPGAs) and combinatorial hops count (C-Hops). The last parameter has an important impact on system performance since it is correlated to the number of times a signal has to be send in the same clock period [20]. In all cases we obtain a partitioning with a maximum combinatorial hop equal to 1. This good partitioning result shows the high quality of the used timing driven partitioner [13].
For the same partitioning result, we applied the routing techniques described in section IV-A:
- OAR: Obstacle Avoidance Routing
- NCR: Negotiated Congestion Routing
In both cases, as shown in figure 10, we try to achieve routing with the smallest multiplexing ratio (Mux-Ratio) and the minimal routing hops (R-Hop). In fact the clock frequency is proportional to a combination of both factors. In our experimentation we used a serial communication clock frequency equal to 250 MHz (multiplexing data clock). Results

TABLE I
BOARD AND GENERATED BENCHMARK CIRCUITS CHARACTERISTICS

	LUTs	RAMLUTs	DSP	RAM	REG
Board	2062080	556758	5184	4320	4124160
CPU_20	143217	6192	2	21	66937
CPU_30	213524	9272	12	33	99588
CPU_50	353697	15432	25	54	164587
CPU_75	510304	20230	20	76	191200
CPU_125	879897	38532	28	130	408712
CPU_150	995750	43280	23	152	449210
CPU_200	1240834	59310	33	207	630411

TABLE II
COMPARISON OF ROUTING STRATEGIES EFFECTS ON PROTOTYPING SYSTEM PERFORMANCE

Benchmark	**Partitioning**				**OAR**			**NCR**			**Gain**
	Occupancy	Used FPGA	Cut	C-Hops	Mux-Ratio	R-Hops	Freq. (MHz)	Mux-Ratio	R-Hops	Freq. (MHz)	
CPU_20	25%	2	450	1	2	0	41	2	0	41	0%
CPU_30	31%	2	620	1	5	1	25	4	1	28	12%
CPU_50	34%	2	945	1	6	2	20	5	1	25	25%
CPU_75	37%	3	1743	1	7	3	18	7	1	21	16%
CPU_125	51%	4	2520	1	10	3	14	10	1	17	21%
CPU_150	58%	5	3700	1	13	2	12	12	1	15	25%
CPU_200	60%	6	5200	1	17	3	10	16	1	12	20%

show the important impact of the NCR iterative routing and its efficiency to improve system performance. The frequency is increased in average by 17% and the impact of NCR is important for highly congested partitioning results (with high Cut count). In fact thanks to its iterative aspect, it avoids easily local minima and reduce the path length from a source FPGA to a destination one. In addition it leads to a good tradeoff between maximum multiplexing ratio and routing hops.

VIII. CONCLUSION

In this paper, we presented the framework of complex and realistic benchmark generator. It is able to design various set of circuits in a very small time using an existing component library. The generation includes the hardware and the software application parts of the circuit. Based on these benchmarks we developed and validated a complete multi-FPGA based prototyping environment. We proposed an innovative extension of PathFinder routing algorithm to route inter-FPGA signals. Compared to common obstacle avoidance algorithms we obtained a significant prototyping system frequency improvement of 17%.

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] N. Pouillon and A. Greiner, URL=https://wwwasim. lip6.fr/trac/dsx/, 2006-2008.

[8] ”Soclib project: Platform for modeling and simulation of integrated systems on chip”, http://www.soclib.fr/.

[9] M. Inc, ”Menta efpga core-ii data sheet brief”, http://www.menta.fr/down/DatasheetBrief-eFPGA-core- II.pdf, Feb. 2009.

[10] Z. Marrakchi and H. Mrabet and U. Farooq and H. Mehrez ”FPGA Interconnect Topologies Exploration” Int. J. Reconfig. Comp. 2009

[11] Synopsys FPGA Synthesis User Guide, 2011.

[12] Xilinx. xst. www.xilinx.com/products/design tools/logic design/ synthesis/xst.htm

[13] www.flexras.com.

[14] www.dinigroup.com/new/dnv6f6pcie.php.

[15] 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.

[16] J.Babb and R. Tessier and M.Dahl and S.Hanono and D. Hoki and A. Aggarwal, ”Logic Emulation with Virtual Wires”, IEEE Trans. Comput.-Aided Des. Inetegr. Circuits Syst., vol. 16, no.6, pp.609-629, June 1997.

[17] R. R. T. Cormen, C. Leiserson. ”Introduction to Algorithms”. MIT Press, Cambridge, MA, 1992.

[18] L. McMurchie and C. Ebeling ”PathFinder: A Negotiation-based Performance-Driven Router for FPGAs”. Proc. FPGA 1995

[19] V. Betz and J. Rose ”VPR: A New Packing Placement and Routing Tool for FPGA research ” International Workshop on FPGA

[20] M.Inagi, Y.Takashima, Y.Nakamura, A. Takahashi ”ILP-based optimization of time-multiplexed I/O assignment for multi-FPGA systems” ISCAS 2008: 1800-1803