2016 IEEE International Parallel and Distributed Processing Symposium Workshops

Efficient Hardware Debugging using Parameterized FPGA Reconfiguration

Alexandra Kourfali
Department of Electronics and Information Systems
Ghent University
Sint-Pietersnieuwstraat 41, B-9000 Ghent, Belgium
Email: alexandra.kourfali@ugent.be

Dirk Stroobandt
Department of Electronics and Information Systems
Ghent University
Sint-Pietersnieuwstraat 41, B-9000 Ghent, Belgium
Email: dirk.stroobandt@ugent.be

***Abstract*—Functional errors and bugs inadvertently introduced at the RTL stage of the design process are responsible for the largest fraction of silicon IC re-spins. Thus, comprehensive functional verification is the key to reduce development costs and to deliver a product in time. The increasing demands for verification led to an increase in FPGA-based tools that perform emulation. These tools can run at much higher operating frequencies and achieve higher coverage than simulation. However, an important pitfall of the FPGA tools is that they suffer from limited internal signal observability, as only a small and preselected set of signals is guided towards (embedded) trace buffers and observed. This paper proposes a dynamically reconfigurable network of multiplexers that significantly enhance the visibility of internal signals. It allows the designer to dynamically change the small set of internal signals to be observed, virtually enlarging the set of observed signals significantly. These multiplexers occupy minimal space, as they are implemented by the FPGA's routing infrastructure.**

I. Introduction

As embedded systems are becoming more complex, errors in the specification, the design and the implementation are unavoidable. Therefore, designers should get proper verification tools to test their design for errors before it is implemented, especially for Application-Specific Integrated Circuits (ASICs), because then errors can not be fixed. Ensuring a design's functional correctness, within time-to-market constraints continues to stand as one of the biggest challenges for today's ASIC design teams. In fact, studies revealed that 35 to 45 percent of the total ASIC development effort is spent on verification [1] and this fraction continues to grow due to the constant increase of chip complexity. Moreover, those studies reveal that debugging consumes about 60 percent of the total verification effort and is the fastest growing component. Thus, verification and particularly hardware debugging has become one of the biggest challenges in designing ASICs.

Software simulation (e.g. Mentor Graphics' ModelSim [2]) has been the standard way to verify and debug circuits, primarily due to its ease of use. For example, designers are able to view the behaviour of any internal signal in their circuit and they can detect design errors, fix them and re-simulate. However, it is often impractical to simulate complete systems, because software simulation scales badly, as it becomes slower when the design is bigger. Furthermore, the complexity of integrated circuits continues to increase, consistent with Moore's Law.

In order to overcome the limitations of software simulation, circuit designers have turned to Field-Programmable Gate Arrays (FPGAs), which are off-the-shelf integrated circuits that can be configured to implement any digital circuit for the simulation of their complete systems. This is called FPGA emulation. It allows early access to hardware tests before the tape-out (the final result of the design cycle for ICs).

FPGA designs operate several orders of magnitude faster than software simulation, while achieving a low-cost per unit and cost less than a fabrication spin [3]. They provide lookup tables or LUTs, equivalent to tens of millions of ASIC gates. Although the faster emulation cycle leads to new opportunities in error diagnosis, verification with FPGA emulation has its own challenges, such as lack of on-chip signal observability.

On-chip signal observability normally is enhanced by instrumentation of the design [4] prior to implementation. These instruments record the values of a subset of signals into embedded memories. These memories are used to record a history of the important signals (or nets), during normal device operation [5]. Then, an engineer can use this information to understand the behaviour of the system. The drawback of this technique is that only a limited amount of such instruments can be inserted due to resource constraints. Therefore, the subset of signals that can be observed is small as well. Another drawback is that the signals have to be selected before compilation. Hence, observing a new subset of signals requires the circuit to be re-instrumented and recompiled, a process that can take hours [6]. Additionally, the insertion of the debug circuitry and the preselection of signals can alter the place and route of the design and can potentially create other problems, such as the user circuit may no longer fit in the FPGA device, or artificial timing limitations caused by the debugging circuitry.

In this paper, we propose ASIC debug acceleration and enhanced internal signal visibility by occasionally changing the subset of observed signals without requiring a complete recompilation. For this, we introduce parameterized configurations (PConf), into the debug cycle of ASICs. A PConf is an FPGA configuration bitstream with some of its bits expressed as Boolean functions of parameters. They can be used to

/16 $31.00 © 2016 IEEE
DOI 10.1109/IPDPSW.2016.95

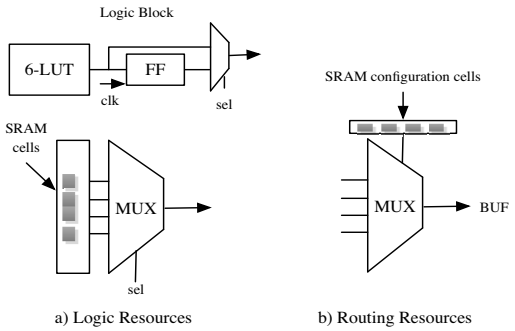

Fig. 1. FPGA's logic and routing resources

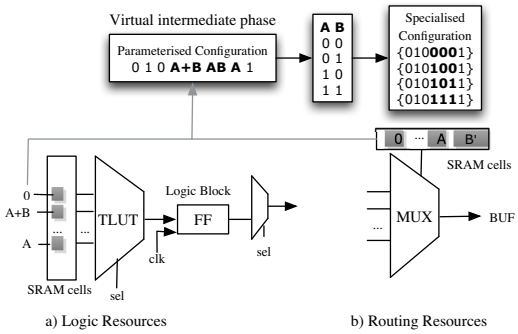

Fig. 2. FPGA's parameterised logic and routing resources

efficiently and quickly generate specialized configuration bitstreams by evaluating the Boolean functions. The specialized bitstreams have slightly different properties and functionalities. With the use of PConf only reconfiguration is needed at debug time and the time consuming recompilation step is avoided. Because reconfiguring even a complete FPGA is very fast compared to recompilation (tens of milliseconds versus minutes to hours), the debug-cycle is sped up significantly. Moreover, the area overhead is reduced. This is achieved by implementing the extra instrumentation (multiplexers) through the use of parameterized reconfiguration of the FPGA's logic and routing infrastructure.

II. Preliminaries

A. Parameterised FPGA Configurations

An FPGA is an array of programmable logic blocks and a configurable routing network. FPGAs implement combinational logic (of up to K variables) by using K-input LUTs. Figure 1 illustrates the logic and routing resources available within an FPGA. This functionality is highly dependent on the programming bitstream that is shifted into the FPGA's SRAM configuration cells.

An application can be parameterised when some of its inputs are infrequently changing compared to the other inputs. Instead of implementing these (parameter) inputs as regular inputs, in our approach these inputs are implemented as constants and the design is optimized for these constants. When the parameter values change, the design is re-optimised (automatically) for the new constant values by reconfiguring the FPGA. This technique can be applied in hardware systems that can be parameterized with parameters that define different circuit instances that can be optimized on the fly by reconfiguring for the current set of parameter values. Lately this tool has been extended and can also support parameterisation of both logic and routing resources [7]. This technique is visualised in Figure 2.

B. FPGA Functional Debug

The modern integrated circuits have come to be extremely large devices which are proving to be difficult to simulate. Researchers are forced to mature their functional verification methodologies to address increasing complexity of ASIC designs. FPGA emulation and prototyping have gained popularity but the tools are still inefficient as they either fail to handle the lack of internal signal visibility or they have several debugging and reimplementing cycles.

Commercial signal capture tools are currently offered by the two major FPGA vendors: Xilinx's ChipScope Pro and Altera's SignalTap II. These tools work by embedding logic analyzer IP (composed of signal probes, trigger monitors, trace buffers and data offload logic) into the user-circuit during regular compilation. A similar, but device-neutral, product is offered by Synopsys as Identify, offering similar functionality. However, although it is possible to modify the trigger conditions (but not the trigger signals) at runtime, changing the signals under observation does require FPGA recompilation. Instrumentation is only done after a failure is observed. FPGA compile times are significant. Another vendor-neutral tool called Certus, allows pre-instrumentation of a large set of interesting signals in the FPGA prior to compilation. Then, during debugging, a small subset of signals can be selected for both observation and triggering. This provides significantly more runtime flexibility to designers than in other tools, but it still requires a set of signals to be preselected for observation before any information about the bugs is known.

Current FPGA trace solutions operate mainly on the design before place and route. These tools will instrument the original user circuit with trace buffers and their connections before mapping, making less resources available for the original design. In real life designs, even for ASICs/multi-core processors, significant memory resources are allocated for tracing, reducing available resources for the design itself, as it is shown in Fig. 3(a). In [10], the authors pre-insert trace buffers into their FPGA ahead of time, and perform low level bitstream modification using incremental techniques to connect them to the desired signals. However, this technique still requires some pre-reservation of FPGA resources, preventing their use by the original design. Also, for every new bug that has to be observed, gaining access to additional signals demands a re-build of the design. The complexity of synthesis and

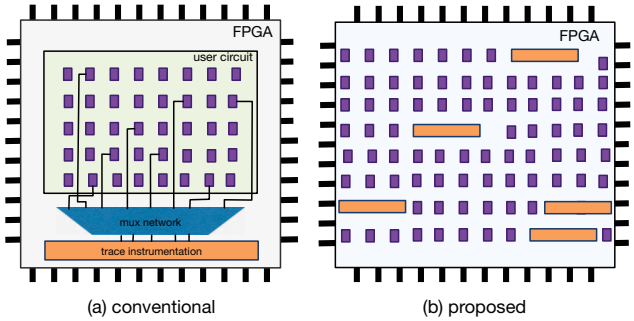

Fig. 3. Figure (a) demonstrates the dedicated area for debugging and figure (b) the proposed approach for elimination of this area and integration of the debugging flow inside the user circuit.

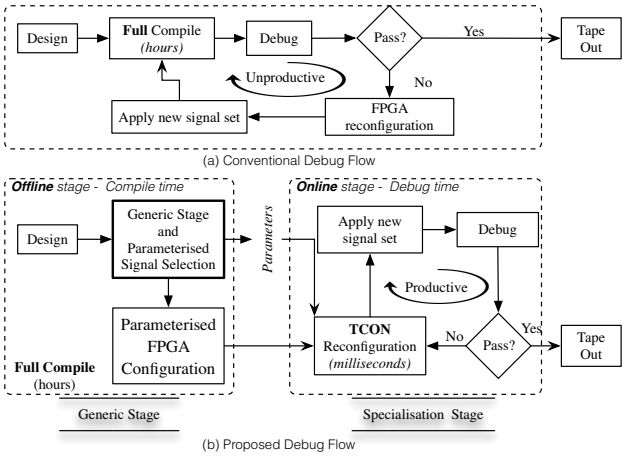

Fig. 4. Proposed debug flow. The two discrete stages offline and online boost time efficiency.

place & route tools can require multiple hours to complete and additional routing stress may cause a design to become unroutable. An improved approach is a debugging workflow that brings good visibility to FPGA-based debugging [11]. A Virtual Overlay Network [12] is used as the basis for the observations, that multiplexes the signals into the trace buffers that are inserted into the free FPGA resources and unnecessary re-spins are avoided. However, this technique relies on spare resources being available which is not always the case.

During FPGA functional debug, when unexpected functionality is observed, a set of signals is selected to be traced. Conventionally, this demands a design recompilation to link the signals to be observed to the trace buffers. As a result, it can take hours to compile large designs [13]. Moreover, the number of signals to be traced during each debugging cycle has to be small and thus you need to use reconfiguration instead of just including all signals. This flow is described in Fig. 4(a).

III. Overview of the Proposed Approach

Our debugging infrastructure is completely integrated inside the normal CAD flow, in order to alter as less as possible the critical path delay and prevent additional routing stress when new signals are to be traced and to offer complete automation of the process. It is outlined in Fig. 4(b). According to this flow, when a designer provides a synthesizable design, the signal parameterisation step sets up the debugging infrastructure. Then, for some selected nets the tool automatically annotates them as parameters, inserts multiplexers that connect each net output to trace buffers and the adapted place and route tool (TPaR) is used to map the FPGA device layout on the technology library of the SRAM-based FPGA to be used. Moreover, it applies the PConf concept and creates a virtual intermediate level of multiplexers. Finally, a generalised bitstream is created that also contains boolean functions. At the end of these steps, before the bitstream is loaded within the FPGA configuration memory, the tool evaluates the boolean function and creates a specialised bitstream. The detailed description of the approach is reported in the following section.

The proposed methodology relies on creating a (virtual, intermediate) bitstream that contains Boolean functions, instead of solely logic-0 and logic-1, according to an optimisation technique used for implementing a PConf on an FPGA. Therefore, by implementing parameterised configurations the extra recompilations are avoided, since only an evaluation of a boolean function is needed, instead of recompilation and/or reconfiguration. Moreover, there are no FPGA resources dedicated to the inserted multiplexers, as it is shown in Fig. 3(b), as these elements are implemented in the PConf. This technique offers signal selection during debug-time. This is done within strict timing constraints and minimal increase in the area overhead.

IV. Parameterised-based Debugging

The proposed approach consists of two phases: the offline phase and the online phase. The generalised stage is created only once, where all signals are multiplexed to trace-buffers. Then, during the online stage, for each debugging cycle the design can be partially reconfigured with specific signals. The signals that are not traced at the same time can share routing resources (based on the parameter settings).

A. The generic stage

The method used to apply our technique enables automatic generation of PConfs starting from parameterised HDL descriptions and is based on the same steps as conventional FPGA tool flows: synthesis, technology mapping, placement and routing [8].

1) Synthesis: The synthesis step can be performed by any tool that is able to synthesise functional blocks to an FPGA flow and communicate directly with ABC, that is a part of the VTR flow [9], a common academic FPGA CAD flow. At this

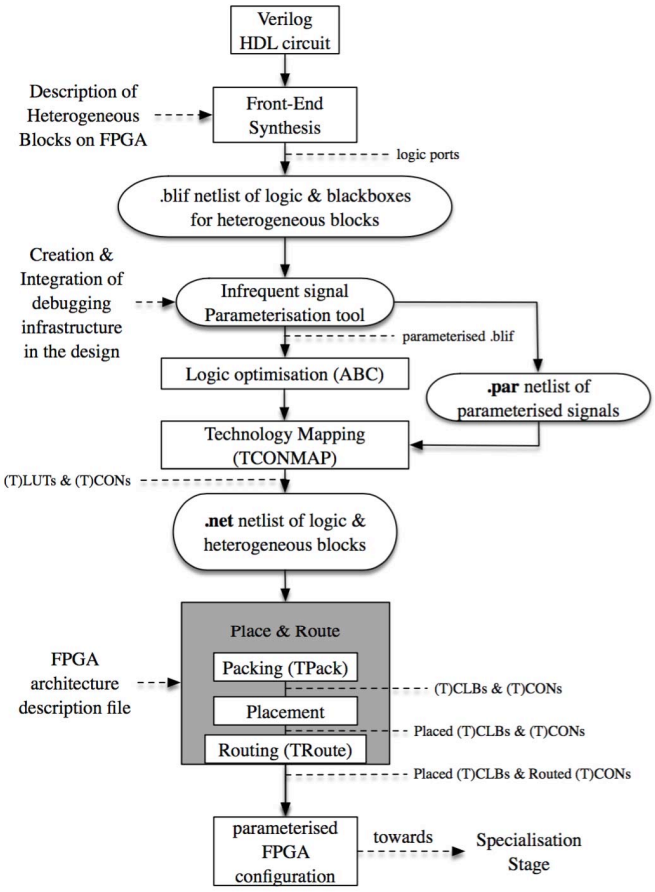

Fig. 5. Schematic of the generic stage of the proposed tool flow.

point the design is ready for signal parameterisation. Figure 5 describes this stage of the tool flow.

2) Signal Parameterisation: The added step in the normal CAD flow can automatically detect and parameterise the internal signals that will be later used for debugging. In more detail, at this stage, extra instrumentation is added, that will be able to assign all signals to trace-buffers. This has to be performed in such a way that after the new modifications, the new description remains synthesizable. The solution is to automatically add multiplexers that will connect the signals to trace-buffers. These signals are annotated as parameters, as they will change (but less frequently than the other signals) depending on the set of signals that will be observed during the online stage. They will indicate whether or not a signal has to be observed in a certain debugging run. The multiplexers are then implemented not in the regular resources but in the FPGA's reconfiguration resources, reducing the overhead significantly (3,5X smaller designs on average). So we basically have almost the same size as for the original circuit but now for an extended circuit with all possible inputs multiplexed to trace-buffers. The entire process is automated, so the designer will not need to manually select signals to be connected to trace-buffers, as the tool handles that. Fig. 6 demonstrates in

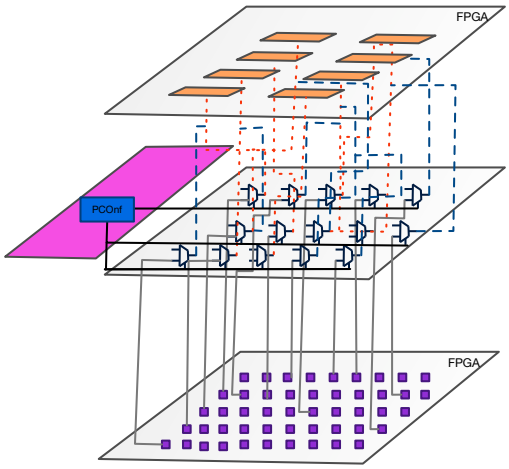

Fig. 6. Demonstration of the separate layers. The user circuit, the parameterised multiplexers and the trace buffers respectively.

different layers how the signal parameterisation is achieved. The bottom layer shows the FPGA and the signals that need to be observed. Then the virtual level adds the infrastructure that multiplexes the signals to trace buffers. We can therefore observe that we no longer need the dedicated FPGA resources that are claimed before implementation, for the multiplexer network and for the trace buffers that is shown in Fig. 3(a).

3) TCON Technology Mapping: During technology mapping, the parameterized Boolean network generated by the synthesis step is not directly mapped onto the resource primitives available in the target FPGA architecture, but intermediately on abstract primitives that introduce and allow the reconfigurability of the logic and routing resources. At this point, the extra multiplexers added to guide the internal signals to trace buffers have their selection bits parameterised into boolean functions and mapped in the virtual abstract primitives.

4) TPaR Placement and Routing: Next, the Tuneable Place and Route tool (TPAR) places and routes the netlist and performs packing, placement and routing with the algorithms TPack, TPlace and TRoute. These algorithms can enable routing of circuits where their routing resources can be reused during the debugging turns and drastically reduce the area usage. At the end of the computationally intensive offline stage the TPaR creates a PConf. Here, a new signal selection during debugging, translates directly into a new evaluation of the function that represents the selected signals. Then it can be reconfigured with Dynamic Partial Reconfiguration (DPR).

B. The specialisation stage

In this stage, the boolean functions are evaluated for a specific parameter value by the Specialized Configuration Generator (SCG) to generate a specialized bitstream. Usually the SCG is implemented on an embedded processor. The

Benchmark	#Gate	Initial	SM	ABC	Proposed (TLUT/TCON)
stereov.	215	208	553	590	190(8/332)
diffeq2	419	422	1719	1819	325(2/712)
diffeq1	582	575	2556	2659	491(4/1065)
clma	8381	4461	23694	23219	7707(1252/7935)
or1200	3136	3084	9769	10958	3004(9/2986)
frisc	6002	2747	11517	11412	5881(2333/4910)
s38417	6096	3462	20695	21040	6204(1495/5597)
s38584	6281	2906	20687	21032	6204(1495/5597)

TABLE I

AREA RESULTS IN #LUTS: THE SECOND COLUMN CONTAINS THE INITIAL DESIGN IN TERMS OF LUTS. THE OTHER COLUMNS CONTAIN THE AREA RESULTS AFTER THE INSERTION OF THE DEBUGGING INFRASTRUCTURE. SM (SIMPLEMAP) AND ABC ARE THE CONVENTIONAL MAPPERS. THE LAST COLUMN DESCRIBES THE RESULTS OF OUR PROPOSED TECHNIQUE.

Benchmark	Golden	SimpleMap	ABC	Proposed
stereov.	4	5	5	4
diffeq2	14	15	15	14
diffeq1	15	15	15	14
clma	11	11	11	11
or1200	27	28	28	27
frisc	14	14	14	14
s38417	7	8	8	7
s38584	7	8	8	7

TABLE II

DEPTH RESULTS. THE FIRST COLUMN DESCRIBES THE LOGIC DEPTH OF THE DESIGN. THE OTHER COLUMNS SHOW THE DEPTH RESULTS AFTER THE ADDITION OF THE DEBUGGING INFRASTRUCTURE AND MAPPING WITH DIFFERENT MAPPERS.

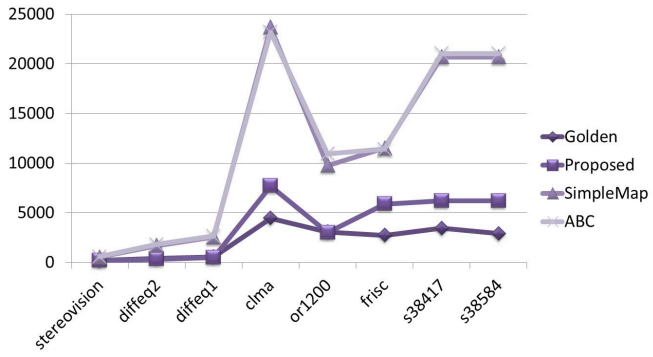

Fig. 7. Area results in terms of look-up tables.

embedded processor is responsible to swap the specialized bitstream into the configuration memory using the HWICAP.

During the specialisation stage (*online phase*), for each debugging cycle the network is partially reconfigured with the exact signals the designer wishes to trace at that specific instance. The multiplexer network added with the signal parameterisation tool is reconfigured with the specialised solution which is evaluated according to the signals that a designer wishes to observe. Here, only the configuration cells of all the routing switch boxes and the connection boxes for the memory resources will be reprogrammed, instead of the full recompilation and/or reconfiguration, as it is the case in related work. Hence, the total wire length is reduced making the proposed technique feasible. However, this online step will be future work, as we first focus on the area gains of the first part of the tool flow.

V. EXPERIMENTAL STUDY

In order to evaluate our proposed method, we have modified PConf with a signal parameterisation step, in order to integrate the debugging infrastructure as part of the flow.

A. Area Usage

The first experiments were conducted with the ISCAS89 and the VTR benchmarks. Starting with the synthesised benchmark (.blif netlist), we run the signal identification and parameterisation part of the flow. This produces a new .blif file and a .par file. The first remains as closely as possible to the original design, while the latter is used to give an indication to the mapper for which signals the PConf should be applied. Then, TCONMap was used to map the design in the abstract logic and reconfiguration resources.

The area results after mapping are shown in Table I. The results indeed indicate that we only need the area for the original circuit, instead of the sum of areas of the initial and the added circuitry. This enables us to include debugging infrastructure without much area overhead, as there was a little area overhead after the insertion of the debugging infrastructure, compared to the initial benchmark. Hence, the adding debugging can be done with low overhead so that (almost) all free space can be used for trace buffers and less is needed for the routing infrastructure to the trace buffers. The debugging infrastructure can then be incrementally added in almost full FPGAs. The area results are shown in Table I. We compare the area results of our method with two conventional tools that are often used in FPGA mapping. The first is SimpleMAP and the second is ABC that is additionally a part of the VTR flow. The area produced with the proposed method is approximately 3,5X smaller than with the conventional mappers, and it can be of similar size with the original design before debugging. These are shown in Fig. 7. If we subtract the original design's LUT utilization from the proposed LUT utilisation and define this as the resources used for the added debugging infrastructure. We can thus observe that there is a high usage of tuneable LUTs and mostly tuneable connections. This is an indicator that the (reconfigured) routing infrastructure is used for our multiplexers, instead of LUTs.

B. Critical Path Delay

Our technique can reduce the critical path delay of the new design with its added functionality for debugging, by reducing the number of LUTs and the routing infrastructure on the critical path. (Table II) shows that the logic depth (inversely related to clock speed) of the design, after adding the extra debugging infrastructure, was either remained the same or reduced, compared to the two conventional mappers. In [14] is shown that with the use of the PConf method, the critical path delay can be up to 8 times smaller compared to the conventional mappers. In fact, it can be of similar size as the original circuit, after the addition of the extra

hardware . Indeed, in this experimental study, after adding the extra routing infrastructure, the critical path delay remains the same compared to the original circuit (without any debugging infrastructure). However, the trade off of the area overhead versus the routing infrastructure that is added has to be investigated further.

C. Timing Impact

1) Compile-time Overhead: In the proposed debugging technique that uses the PConf method, there are a lot of multiplexers that have to be implemented in the routing infrastructure. Even though the routing is only used when the parameters are activated, many routing resources are needed to make this possible. This has a large impact on the parameterised router (taking a lot of time to find a suitable routing). It can also lead to a shortage of routing wires in heavily congested regions where many signals are chosen to be debugged. Early experiments indicate that with the use of the PConf technique we have 3 times less cables (5316 with parameterised resources Vs 15699 for normal LUT architecture for small designs), and runtimes can be up to 3 times faster for place and route for the same designs. Moreover we can have up to 4 times less CLBs. However, our router will need further adaptations to support the congested regions in order to handle larger designs.

2) Run-time Overhead: The runtime overhead depends on the number of times the emulator needs to be reconfigured and on the time to evaluate the PConf and to reconfigure the bits that changed. The time overhead can be expressed as the single specialization time (for specializing the FPGA once) multiplied by the number of times a new signal set will be activated. The evaluation time is used to evaluate the Boolean functions in the parameterized configuration produced by the offline generic stage of the tool flow (maximum 50 μs). Thus, each parameterised configuration can be 3 orders of magnitude faster than a full reconfiguration (176 milliseconds for a Xilinx Virtex-5 FPGA). Also, assuming the FPGA design runs at 400 MHz (which is quite fast for an FPGA implementation) and the debug loop in Figure 4(b) can be executed in 4 clock ticks (which requires a fully pipelined design), the 50 μs overhead corresponds with the time needed to perform 5000 debugging turns on the FPGA fabric. So the overhead is only amortised if significantly more debugging cycles than 5000 are performed before a new signal needs to be observed. This is a reasonable number for large designs. So for larger designs, the overhead becomes smaller relative to the debugging turn.

VI. Conclusion and Future Work

A low overhead debugging method is proposed. The main (parameterised) debugging infrastructure is presented, which is meant for both emulation approaches (for ASIC verification) and on-line in field debugging approaches (for FPGA design verification) and it includes increasing design observability. This infrastructure is embedded within the circuit implementation and is only invoked when a debugging parameter is set. Therefore, this infrastructure is always present (and hence a recompilation for new signals to be observed is never needed) but does not require much additional area. Hence, thanks to the fact that there is low overhead over the original implementation, we can add the debugging functionality almost for free.

In future, the reconfiguration time of the place and routing solution produced with the PConf method will be further investigated to handle the congested routing due to our excessive use of the multiplexers that are implemented in the routing resources for the debugging infrastructure. Moreover, the implementation of a critical signal selection technique is planned, in order to reduce the parameters that are automatically produced by the tool flow. Thus, we will be able to limit the compile time overhead and the area overhead even further.

Acknowledgment

The first author is sponsored by a Ph.D. grant of the Flemish Fund for Scientific Research (FWO). This work was supported by the European Commission in the context of the H2020 FETHPC EXTRA project (#671653).

References

[1] M. Abramovici, P. Bradley, and K. Dwarakanath, "A reconfigurable design-for-debug infrastructure for socs," in *Design Automation Conference, 2006 43rd ACM/IEEE*, 2006.

[2] Mentor Graphics, "ModelSim User's Manual," 2012.

[3] Hutchings, Brad L. and Keeley, Jared, "Rapid Post-Map Insertion of Embedded Logic Analyzers for Xilinx FPGAs," in *Field-Programmable Custom Computing Machines (FCCM), 2014 IEEE 22nd Annual International Symposium on*, 2014.

[4] Xilinx, "Chipscope pro software and cores (v14.3)," 2012.

[5] E. Hung, A. S Jamal and S. Wilton, "Maximum flow algorithms for maximum observability during FPGA debug," in *Field-Programmable Technology (FPT), 2013 International Conference on*, 2013.

[6] S. Chin and S. Wilton, "An analytical model relating fpga architecture and place and route runtime," in *Field Programmable Logic and Applications, FPL 2009. International Conference on*, 2009.

[7] E. Vansteenkiste, Brahim Al Farisi, Karel Bruneel, and Dirk Stroobandt, "TPAR: Place and route tools for the dynamic reconfiguration of the FPGA's interconnect network, " in *Transactions on Computer-Aided Design of Integrated Circuits and Systems, IEEE*, 2014.

[8] Karel Heyse, Karel Bruneel, and Dirk Stroobandt, " Mapping logic to reconfigurable fpga routing, " in *Field Programmable Logic and Applications (FPL), 2012 22nd IEEE International Conference on*, 2012.

[9] J. Luu, J. Goeders, M. Wainberg, A. Somerville, T. Yu, K. Nasartschuk, M. Nasr, S. Wang, T. Liu, N. Ahmed, K. B. Kent, J. Anderson, J. Rose and V. Betz, "VTR 7.0: Next Generation Architecture and CAD System for FPGAs," in *ACM Trans. Reconfigurable Technol. Syst.*, 2014.

[10] Graham, P. and Nelson, B. and Hutchings, B., "Instrumenting Bitstreams for Debugging FPGA Circuits," in *Field-Programmable Custom Computing Machines, 2001. FCCM '01. The 9th Annual IEEE Symposium on*, 2001.

[11] E. Hung and S. Wilton, "Scalable signal selection for post-silicon debug," in *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 21, 2013.

[12] E. Hung and S. J. Wilton, "Towards simulator-like observability for fpgas: a virtual overlay network for trace-buffers," in *Proceedings of the ACM/SIGDA international symposium on Field programmable gate arrays, FPGA '13*, 2013.

[13] Z. Poulos, Yu Shen Yang, Anderson, J., Veneris, A. and Bao Le, "Leveraging reconfigurability to raise productivity in FPGA functional debug," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2012*, 2012.

[14] Alexandra Kourfali and Dirk Stroobandt, "Test set generation almost for free using a run-time FPGA reconfiguration technique," in *Test Symposium (LATS), 16th IEEE Latin-American,*, 2015.