Utilizing Dynamic Partial Reconfiguration to Reduce the Cost of FPGA Debugging

Islam Ahmed1, Ahmed Kamaleldin2, Hassan Mostafa2,3, Ahmed Nader Mohieldin2

1IC Verification Solutions, Mentor Graphics, a Siemens Business, Cairo, Egypt
2Electronics and Communications Engineering Department, Cairo University, Giza 12613, Egypt
3Nanotechnology Program at Zewail City of Science and Technology, Cairo, Egypt
{islam_ahmed@mentor.com, ah.kamal.ahmed@gmail.com, hmostafa@uwaterloo.ca, anader2000@yahoo.com}

Abstract—Debugging of Field-Programmable Gate Arrays (FPGAs) is a difficult task due to the limited access to the internal signals of the design. Embedded logic analyzers enhance the signal observability for FPGAs. These analyzers are implemented on the FPGA resources and they use the embedded memory blocks as trace buffers, so a limited number of signals can be observed using these analyzers due to resources constraints. Changing the traced set of signals requires re-synthesis, placement and routing of the whole design. In this paper, we propose a new methodology for FPGA debugging to change dynamically the set of signals to be observed at runtime, and consequently minimize the time required for debugging. The proposed methodology utilizes the Dynamic Partial Reconfiguration (DPR) technique to dynamically switch between different sets of signals. DPR creates a reconfigurable module (RM) to route each set of signals to an embedded logic analyzer. We demonstrate the proposed approach using Xilinx FPGA tools, finding that changing the set of signals to be observed requires only few milli-seconds to re-program the reconfigurable region (RR). The area overhead of the proposed methodology is lower than other traditional methods of using multiplexers as the DPR allows the routing module to only use buffers to connect a set of signals to the embedded logic analyzer.

I. INTRODUCTION

Verification is one of the most challenging tasks in the Integrated Circuits (ICs) development process. Any uncaught bugs or errors during the design and verification phases can cause re-spins for silicon IC. Studies revealed that about half of designer’s effort is spent on functional verification [1]. With the increased complexity and size of the designs, traditional functional verification methodologies like RTL simulation are no longer sufficient to uncover bugs and errors in the design because some real world interactions only show up when implemented on hardware. Simulation also runs at lower speeds than real hardware execution [2], [3] which makes the thorough analysis of large designs infeasible.

Reconfigurability of Field-Programmable Gate Arrays (FPGAs) attracts designers to do prototyping for their systems. FPGAs can run at higher speeds than that of simulation, and will catch bugs and errors that cannot be caught in simulation like system timing issues. Debugging design and system integration issues on FPGAs is a difficult task due to the limited access to internal signals, designer can only observe the signals connected to the FPGA output pins. Embedded logic analyzers are used to provide visibility for internal signals inside the FPGA [4], [5], [6]. These analyzers are implemented on the FPGA resources, they use embedded memory blocks as trace buffers. Designers use the Joint Test Action Group (JTAG) port to access the analyzer and the recorded data can be replayed on a Personal Computer (PC). The traditional design and debug flow for FPGAs is shown in Fig. 1. The major disadvantage of using embedded logic analyzers is that the observed signals that are connected to the trace buffer of the embedded logic analyzer are selected before the user design is synthesized, placed and routed. In order to change the set of observed signals, it will require the recompilation of the FPGA design flow. Also, the debug circuitry added in the design consumes a part of the FPGA resources, so the Design Under Test (DUT) may no longer fit in the FPGA device. The amount of resources required for debugging is directly proportional to the number of selected signals to be observed.

Dynamic Partial Reconfiguration (DPR) for FPGAs allows a part of the FPGA to be configured while the rest of the logic keeps operating. It allows the implementation of complex designs that have multiple operating modes like Software Defined Radio (SDR) applications within reasonable area on the FPGA. In DPR, the design consists of a number of Reconfigurable Modules (RMs), each module has a number of modes that are changed during runtime according to the system operating modes. A Reconfigurable Region (RR) is a location on the FPGA in which the RM is allocated on. An example for DPR system is shown in Fig. 2, it has five configuration modes, each configuration has four RMs: ModuleA, ModuleB, ModuleC and ModuleD, each with four modes: Mode1, Mode2, Mode3 and Mode4. DPR extends the design flexibilty through mapping of multiple RMs to the same

Fig. 1. Design and debugging flow for FPGAs.
physical RR, which reduces the design cost and the resource usage. In the example of Fig. 2, we will have 4 RRs on the FPGA, each RR is used for a unique RM.

Our approach utilizes DPR on FPGAs to alleviate the issues of using embedded logic analyzers by dividing the large number of all potential signals for debugging. For every mode of the RM, a set of signals is connected to the probes of the embedded logic analyzer. The methodology can be extended to use the output pins of the FPGA for observing the selected signals instead of the embedded logic analyzer, by connecting the outputs of the RM to the output pins of the FPGA. The changes in the connections of the signals sets to the analyzer are done at runtime. So, the proposed methodology avoids the recompilation of the whole FPGA flow by changing the observed signals during runtime, and also it controls the size of the logic analyzer by controlling the number of its probes without affecting the observability of potential debugging signals, as they are still observable by changing the mode of the RM at runtime.

This paper is organized as follows: Section II shows the related work for improving FPGA debugging capabilities, and Section III presents the proposed FPGA debugging methodology using DPR. Section IV demonstrates the experimental results. Finally, Section V draws the paper conclusion.

II. RELATED WORK

Several works have proposed techniques to enhance the debugging of FPGAs using scan-based or trace-based techniques. In [7], a scan-based technique is proposed to connect all the Flip-Flops (FFs) in sequence by using the soft-logic of the FPGA. This technique has a high area overhead due to the usage of the soft-logic to implement the scan-chains in the design.

A bitstream modification technique is presented in [8] to modify the bitstreams within tens of seconds to minutes. This can reduce the time spent in debugging the design, and decrease design’s time to market. But, when the selected set of signals for tracing is changed, re-routing needs to be performed which can significantly affect the design’s time to market. Software-like debug features are presented in [9] such as watchpoints and breakpoints to enhance debug capability in reconfigurable platforms. But, any change in watchpoints or breakpoints needs recompilation of designs.

In [10], a new methodology is proposed to permit a large number of internal signals to be traced for an arbitrary number of clock cycles using a limited number of external pins. It operates without the need of iterative executions of the design re-synthesis, placement and routing tools. This is achieved by inserting a Multiplexer (MUX) into the design implemented on the FPGA, with the MUX inputs being all the signals that designer potentially needs to trace. Then, the select signals of the MUX are controlled by manipulating the bitstream of the design to select different signals to be traced. The disadvantage of this methodology is the area overhead of the MUX, and the need to re-program the whole FPGA for any change in the selected signals to be traced.

This section presents a new approach to enhance the observability of FPGA designs for debugging. The traditional debugging flow for FPGA designs is shown in Fig. 1, the design is synthesized, placed and routed on the target FPGA, then the generated bitstream is used to program the FPGA. During the testing, if an issue is caught, a set of signals is selected to be observed by an embedded logic analyzer, or by routing them to the available output pins. In that case, the designer needs to repeat the FPGA design flow from synthesis to FPGA programming which is time consuming. Additionally, observing a large number of signals is not feasible in the traditional debugging flow because of the limited resources of the FPGA either for the memory blocks and look-up tables (LUTs) in case of the embedded logic analyzer, or for the output pins in case these pins are used for debugging. This forces the designer to repeat the FPGA design flow multiple times in order to observe different sets of signals to debug different faulty scenarios. For the rest of the paper, we will assume that an embedded logic analyzer is being used for debugging for simplicity, the proposed approach and the results presented are still applicable for using the output pins for debugging, the only difference is to replace the number of analyzer’s probes by the number of the output pins available for debugging.

A new approach for FPGA debugging is presented in this work to overcome the limitations of the traditional FPGA debugging flow. This approach allows the designer to switch between the signals at runtime without need to repeat the FPGA design flow. This is achieved by inserting one RM in the design to switch between the signals to be observed. This RM is implemented on an RR on the FPGA. All the potential signals to be observed are connected as inputs to this module. The outputs of the RM are connected to the embedded logic analyzer or the debug output pins. Fig. 3 shows the connections of the RM. Depending on the available resources on the FPGA, the number of modes of the RM is decided. For number of signals to be observed $Nsigs$ and number of probes $Nprobes$ for the embedded logic analyzer, the number of modes of the RM is $Nsigs/Nprobes$.

For each mode of the RM, a set of signals is connected to the probes of the embedded logic analyzer. An example for 4-inputs and 2-outputs case is shown in Fig. 4. For each mode, a set of signals is routed to the output while the others

<table>
<thead>
<tr>
<th>Config</th>
<th>ModuleA_Mode1</th>
<th>ModuleB_Mode2</th>
<th>ModuleC_Mode3</th>
<th>ModuleD_Mode4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Config</td>
<td>ModuleA_Mode1</td>
<td>ModuleB_Mode2</td>
<td>ModuleC_Mode3</td>
<td>ModuleD_Mode4</td>
</tr>
<tr>
<td>Config</td>
<td>ModuleA_Mode1</td>
<td>ModuleB_Mode2</td>
<td>ModuleC_Mode3</td>
<td>ModuleD_Mode4</td>
</tr>
<tr>
<td>Config</td>
<td>ModuleA_Mode1</td>
<td>ModuleB_Mode2</td>
<td>ModuleC_Mode3</td>
<td>ModuleD_Mode4</td>
</tr>
<tr>
<td>Config</td>
<td>ModuleA_Mode1</td>
<td>ModuleB_Mode2</td>
<td>ModuleC_Mode3</td>
<td>ModuleD_Mode4</td>
</tr>
<tr>
<td>Config</td>
<td>ModuleA_Mode1</td>
<td>ModuleB_Mode2</td>
<td>ModuleC_Mode3</td>
<td>ModuleD_Mode4</td>
</tr>
</tbody>
</table>

Fig. 2. An example of DPR design with five modes of configuration and four reconfigurable modules per configuration.

III. AN APPROACH FOR FPGA DEBUGGING USING DYNAMIC PARTIAL RECONFIGURATION

Section IV demonstrates the experimental results and number of probes for debugging. The traditional debugging flow for FPGA designs is shown in Fig. 1, the design is synthesized, placed and routed on the target FPGA, then the generated bitstream is used to program the FPGA. During the testing, if an issue is caught, a set of signals is selected to be observed by an embedded logic analyzer, or by routing them to the available output pins. In that case, the designer needs to repeat the FPGA design flow from synthesis to FPGA programming which is time consuming. Additionally, observing a large number of signals is not feasible in the traditional debugging flow because of the limited resources of the FPGA either for the memory blocks and look-up tables (LUTs) in case of the embedded logic analyzer, or for the output pins in case these pins are used for debugging. This forces the designer to repeat the FPGA design flow multiple times in order to observe different sets of signals to debug different faulty scenarios. For the rest of the paper, we will assume that an embedded logic analyzer is being used for debugging for simplicity, the proposed approach and the results presented are still applicable for using the output pins for debugging, the only difference is to replace the number of analyzer’s probes by the number of the output pins available for debugging.

A new approach for FPGA debugging is presented in this work to overcome the limitations of the traditional FPGA debugging flow. This approach allows the designer to switch between the signals at runtime without need to repeat the FPGA design flow. This is achieved by inserting one RM in the design to switch between the signals to be observed. This RM is implemented on an RR on the FPGA. All the potential signals to be observed are connected as inputs to this module. The outputs of the RM are connected to the embedded logic analyzer or the debug output pins. Fig. 3 shows the connections of the RM. Depending on the available resources on the FPGA, the number of modes of the RM is decided. For number of signals to be observed $Nsigs$ and number of probes $Nprobes$ for the embedded logic analyzer, the number of modes of the RM is $Nsigs/Nprobes$.

For each mode of the RM, a set of signals is connected to the probes of the embedded logic analyzer. An example for 4-inputs and 2-outputs case is shown in Fig. 4. For each mode, a set of signals is routed to the output while the others
remain unconnected, so buffers only will be used to do this connection, and LUTs of the FPGA will not be used. This is a major advantage for using this approach because the area will be as minimum as possible when compared with other approaches that use MUXes to switch between the signals sets like the proposed approach in [10]. For each mode, a partial bitstream is generated and it will be used to re-program the RR at runtime. Partial bitstreams are generated during the DPR design flow and are saved into an external memory. The size of a partial bitstream is directly proportional to the size of the RR [11]. Since, the area consumed by each mode of the RM is very small because it only uses buffers, the size of the partial bitstream will be small, and consequently the reconfiguration will require few milliseconds to re-program the RR. The small reconfiguration time is a major advantage for the proposed approach in this work when compared with the traditional FPGA debugging flow as it avoids re-compilation, and also when compared with other approaches which do modifications in the bitstream then re-program the whole FPGA as in [10]. The proposed FPGA debugging flow is shown in Fig. 5.

IV. EXPERIMENTAL RESULTS

The experiment aims to study the utilization of DPR to reduce the cost of FPGA debugging in terms of area overhead of the RM, time of reconfiguration (i.e. time needed to switch between different sets of traced signals), and the usability of the FPGA debugging flow.

A. System Implementation and Setup

The experimentation is carried out using Xilinx Zynq XC7Z020LG484-1 FPGA and tested with a ZC702 board [12]. The DPR flow has been carried out using Xilinx Vivado tool. The complete system is developed as shown in Fig. 6. The Zynq FPGA device consists of two parts: i) The Programmable Logic (PL) and ii) The Processing System (PS) part. The PL part contains: 1) the Design Under Test (DUT) that is used as a test case to evaluate the proposed FPGA debugging flow, 2) The reconfigurable partition region which is used to host the RM modes of the debugging interfaces, 3) The embedded logic analyzer (ILA) is used to capture the observed signals and send them to an external PC. The proposed flow can be applied using the output pins of the FPGA instead of the embedded logic analyzer, so we are interested in calculating the metrics for the RM to compare it with the area-optimized MUX presented in [10]. The DPR process is done using the serial JTAG external configuration port to load the partial bitstreams of the debugging modes interfaces from an external PC to the FPGA configuration memory with a data rate of 66 Mb/s [11].

In our experiments, we are using the same DUT setup as in [10] to compare the results of the two proposals against each other. The DUT was modified to connect the traced signals to the proposed RM. Xilinx’s attribute, keep, was used to prevent the removal of these signals during optimization. In the following subsections, the notation, m-w, represents the tracing setting where m signals are candidates for tracing and w signals are traced concurrently.

B. Area Overhead

The area overhead of the proposed RM for 6 different tracing settings is shown in Table I. It is found that the area overhead is directly proportional to the number of signals observed concurrently (i.e. those connected to the embedded logic analyzer), it is not changing with the number of candidates signals for debugging. Xilinx Vivado’s place and route tool creates a partition pin for every input port of the RM. Partition pins are physical connection between static logic and reconfigurable logic, they are automatically created for all Reconfigurable Partition ports [11]. The partition pins are implemented on the interconnect resources of the RR on the FPGA. In the following table, the notation, m-w, represents the tracing setting where m signals are candidates for tracing and w signals are traced concurrently.
Table I reports the area overhead for the proposed structure in [10] in terms of 4-input LUTs. This overhead is calculated by multiplying the number of Adaptive Logic Modules (ALMs) by two, this is because each ALM in an Altera Stratix III device can contain two 4-input LUTs [10]. The area overhead of our proposed approach is smaller than that of [10]. This is expected because two 64:1 MUXes are needed for the 128-2 trace setting in [10], while our DPR approach will only use two 1-input LUTs for the 128-2 trace setting.

### Table II

<table>
<thead>
<tr>
<th>Trace Setting</th>
<th>128-2</th>
<th>128-4</th>
<th>128-8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average Number of 4-input LUTs</td>
<td>50</td>
<td>50</td>
<td>50</td>
</tr>
<tr>
<td>Trace Setting</td>
<td>256-2</td>
<td>256-4</td>
<td>256-8</td>
</tr>
<tr>
<td>Average Number of 4-input LUTs</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
</tbody>
</table>

### C. Time for Changing the Traced Signal Set

The time needed to switch between different signals sets is equivalent to the reconfiguration time of the RR. The reconfiguration time of the RR is calculated as:

$$t_{reconfig} = \frac{size_{pbs}}{bit\text{-}rate_{jtag}}$$

Where $t_{reconfig}$ is the time to switch between a traced set of signals to another, $size_{pbs}$ is the size of the partial bitstream file, and $bit\text{-}rate_{jtag}$ is the bit rate of the JTAG port which is used to re-program the RR on the FPGA. For the setup considered in this work, $bit\text{-}rate_{jtag}$ is 66 Mb/s [11], and the size of the partial bitstream is 30 KB. So, the time to switch between a traced set of signals to another is 3.63 ms.

The time needed to cover all the signals sets is calculated as:

$$t_{total\text{-}sw} = N_{modes} \times t_{reconfig}$$

Where $t_{total\text{-}sw}$ is the time needed to cover all the signals sets of the candidate signals for debugging, $N_{modes}$ is the number of modes of the RM that are implemented on the RR, and $t_{reconfig}$ is the time needed to reconfigure the RR as calculated in (1). Table III shows the total switching time required to trace all the signals sets. The switching time for the proposed debugging flow is much less than that of [10]. In [10], the bitstream should be manipulated to change the select signals of the area optimized MUX, then the whole FPGA needs to be re-programmed. The authors of [10] report that it takes seconds to change the traced signal set. Similarly, the switching time of our proposed flow is much faster than the switching time of the traditional debugging flow which requires minutes for the re-compilation of the FPGA design flow. Another advantage of the proposed flow, is that the switching of the signals sets can be done at runtime, unlike other methodologies which require the whole FPGA to be re-programmed.

### V. Conclusion

Debugging of FPGA devices is a difficult task due to the limited access to the internal signals in the design. Traditional debugging flow requires re-compilation of the FPGA design flow in order to change set of observed signals either through embedded logic analyzer or output pins of the FPGA. This work presents a new technique to use the DPR design flow to reduce the cost of the debugging on FPGA devices. The new technique has a small area usage as the DPR flow allows the switching between signals to use buffers only to wire a selected signal set to the embedded logic analyzer or the FPGA output pins. The FPGA reconfiguration to switch the traced signal set requires milli-seconds to program the RR on the FPGA.

### REFERENCES