Automatic Clock Domain Crossing Verification Flow For Dynamic Partial Reconfiguration

Islam Ahmed[1], Hassan Mostafa[2,3], Ahmed Nader Mohieldin[2]
[1]IC Verification Solutions, Mentor Graphics, a Siemens Business, Cairo, Egypt
[2]Electronics and Communications Engineering Department, Cairo University, Giza 12613, Egypt
[3]Nanotechnology Program at Zewail City of Science and Technology, Cairo, Egypt
{islam_ahmed@mentor.com, hmostafa@uwaterloo.ca, anader2000@yahoo.com}

***Abstract*—Dynamic Partial Reconfiguration (DPR) on Field Programmable Gate Arrays (FPGAs) allows some of the logic to be configured while the rest of the logic keeps operating. This kind of designs are called Dynamically Reconfigurable System (DRS) designs, they can operate in multiple modes. The verification of the DRS designs is a complicated task due to the need to verify all the modes of the designs, and the lack of Computer Aided Design (CAD) tools support for DRS designs. In this paper, we propose an automatic Clock Domain Crossing (CDC) verification flow for DRS designs. A Perl utility is implemented which automates the generation of the designs files for each operating mode of the design, generates the script to run CDC analysis on the design, runs a CDC analysis tool, and collates the results in a user-friendly representation for debugging.**

I. Introduction

Dynamic Partial Reconfiguration (DPR) on Field Programmable Gate Arrays (FPGAs) allows some of the logic to be configured while the rest of the logic keeps operating [1][2]. It allows the implementation of complex circuits such as Software Defined Radio (SDR) and Internet of Things (IoT) applications within a reasonable area on the FPGA, and consequently, the power consumption of the circuit is reduced. Recent FPGA families support the implementation of DRS through the Dynamic Partial Reconfiguration (DPR) technique.

In DPR, the design consists of a number of Reconfigurable Modules (RMs), each module has modes that are changed at runtime according to the system operating modes. A Reconfigurable Region (RR) is a location on the FPGA in which the reconfigurable module is implemented on. An example for DPR system is shown in Fig. 1, it has five configuration modes: *Config1*, *Config2*, *Config3*, *Config4* and *Config5*. Each configuration has four RMs: *ModuleA*, *ModuleB*, *ModuleC* and *ModuleD*, each with four modes: *Mode1*, *Mode2*, *Mode3* and *Mode4*. DRS designs extend the design flexibility through mapping of multiple RMs to the same physical RR. This reduces the design cost and the resources usage. In the example of Fig. 1, the design will have 4 RRs on the FPGA, each RR is used for a unique RM.

Most complex recent designs have more than one clock, and many of these clocks are asynchronous. For these designs, the logic clocked by each asynchronous clock forms the clock domain for the clock. Problems arise from signals that connect logic in different clock domains. Signals that cross clock domain boundaries must be properly synchronized, and they must obey all relevant transfer protocols. If any CDC signal does not hold steady during the setup and hold time of its receiving register, then the register can become metastable, and its output can settle at random to a value that is different from the Register Transfer Level (RTL) simulated value, an example is shown in Fig. 2.

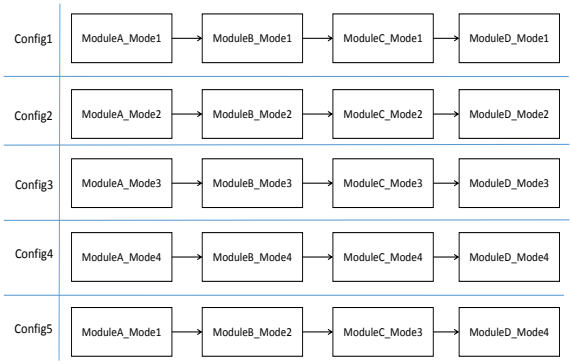

Fig. 1. An example of DPR design with five modes of configuration and four RMs per configuration.

Such metastability issues can cause functional errors in the design. CDC verification [3] of DRS designs is a complicated task due to the need of verifying every operating mode of the design to make sure no metastability issues can occur in the design. Currently, there are no Computer Aided Design (CAD) tools that support the CDC verification of DRS designs. As example in Fig. 1, designers should verify all the configuration modes of the design, to make sure any CDC signals between adjacent modules are properly synchronized. If CDC errors are not verified and tackled early in the design cycle, they may cause functional errors later in the synthesis and place & route phases which may waste the designer's time to repeat the design cycle after fixing the CDC errors.

In this paper, we propose a new automated flow for CDC verification of DRS. A Perl utility is implemented to 1) automate the generation of the Register Transfer Level (RTL) representation of every operating mode of the reconfigurable system, 2) generate the run scripts to run a commercial CDC tool for every mode, 3) invoke the run for CDC analysis and 4) collates the result for every mode and report it to the user.

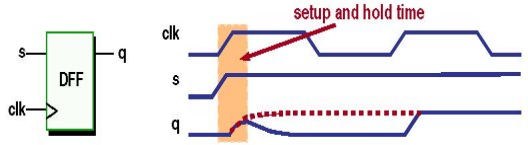

Fig. 2. Example for a metastability issue caused by CDC signal.

II. Background

The verification of DRS designs is still an open question. The lack of CAD tools that understand the dynamic nature of these designs forces the designers and verification engineers to innovate and implement their own verification methodologies. Several works have proposed techniques for simulation based verification of DRS designs, and verification of issues that may arise before, during, and after reconfiguration of some part of the design.

The Dynamic Circuit Switch (DCS) method [4] adds artifacts in the RTL code of the DRS design for simulation purposes only to switch between hardware tasks, it improves the simulation accuracy of DRS designs in various aspects. But, using this method cannot detect bugs introduced by bitstream transfer and the module swapping in DRS designs. ReChannel [5] [6] is an open source SystemC library which models DPR, it was extended in [7]. ReChannel adds new SystemC classes to encapsulate reconfiguration operations such as module swapping. The extension of ReChannel [7] proposed new classes to capture the reconfiguration details and proposed a modeling methodology to assist functional verification of DPR designs at behavioral, Transaction Level Modeling (TLM) and RTL levels. However, DCS and ReChannel do not accurately verify the design undergoing reconfiguration since the bitstream traffic is not simulated.

ReSim [8] is a reusable library which uses simulation-only bitstreams to hide the physically dependent details of DPR designs. It models the bitstream traffic and the reconfiguration process of DPR. It supports the cycle-accurate RTL simulation of the DRS design before, during and after reconfiguration. In [9], a framework is proposed to verify the reconfiguration logic using Assertion Based Verification (ABV) [10].

The existing work in literature focuses on simulation-based functional verification of DRS designs, there are more advanced verification topics that are not still addressed for DRS designs such as CDC verification, reset verification, power-aware verification, formal verification and runtime verification. In this paper, we introduce a framework for CDC verification for DRS designs.

III. CDC Verification Flow For DRS Designs

The proposed CDC Verification flow is shown in Fig. 3. A Perl utility is implemented to automate the flow. The inputs for the utility are: 1) RTL files of RMs modes, 2) RTL wrapper for DRS design and 3) Comma Separated Values (CSV) file to define the configuration modes of the design. In a typical DPR design flow, the RTL files of the RMs modes and the wrapper of the DRS design should be created by the designer, so there is no extra effort needed for the creation of these files to use the proposed CDC verification flow. Following is an example for Verilog RTL code which defines two modes of the RM (ModuleA) in Fig. 1:

```
module ModuleA_mode1 (input wire in1, in2, a_rst, clk1,
                      output reg out1);
  always @(posedge clk1, posedge a_rst) begin
    if (arst) out1 <= 1'b0; else out1 <= in1 | in2; end
endmodule
```

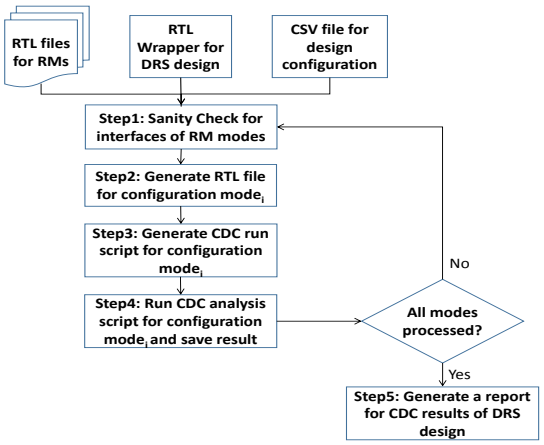

Fig. 3. Proposed CDC verification flow.

```
module ModuleA_mode2 (input wire in1, in2, a_rst, clk1,
                      output reg out1);
  always @(posedge clk1, posedge a_rst) begin
    if (arst) out1 <= 1'b0;  else out1 <= in1 & in2; end
endmodule
```

The following Verilog RTL mode shows an example for the wrapper of the DPR design example in Fig. 1:

```
module RR1(input wire in1, in2, a_rst, clk1, output out1);
// Empty. A mode for ModuleA will be instantiated here
endmodule
module RR2(input wire in1, in2, a_rst, clk1, output out1);
// Empty. A mode for ModuleB will be instantiated here
endmodule
module RR3(input wire in1, in2, a_rst, clk1, output out1);
// Empty. A mode for ModuleC will be instantiated here
endmodule
module RR4(input wire in1, in2, a_rst, clk1, output out1);
// Empty. A mode for ModuleD will be instantiated here
endmodule
module DRS1 (input wire in1, in2, in3, in4, in5,
             a_rst, clk1, clk2, output wire out1);
  wire A_out1, B_out1, C_out1, D_out1;
  RR1 ModuleA_inst (in1, in2, a_rst, clk1, A_out1);
  RR2 ModuleB_inst (in3, A_out1, clk1, B_out1);
  RR3 ModuleC_inst (in4, B_out1, clk2, C_out1);
  RR4 ModuleD_inst (in5, C_out1, clk2, D_out1);
  assign out1 = D_out1;
endmodule
```

The CSV file is needed to define the configuration modes of the design, so that the utility can know how many RRs in the design and what are the RMs mapped to a specific RR. The following CSV file is an example for the DPR design in Fig. 1:

```
1  RR,RR1
2  RR,RR2
3  RR,RR3
4  RR,RR4
5  RM,ModuleA,{ModuleA_Mode1,ModuleA_Mod2,ModuleA_Mode3,
       ModuleA_Mode4}
6  RM,ModuleB,{ModuleB_Mode1,ModuleB_Mod2,ModuleB_Mode3,
       ModuleB_Mode4}
7  RM,ModuleC,{ModuleC_Mode1,ModuleC_Mod2,ModuleC_Mode3,
       ModuleC_Mode4}
8  RM,ModuleD,{ModuleD_Mode1,ModuleD_Mod2,ModuleD_Mode3,
       ModuleD_Mode4}
9  ConfigMode,Config1,{{RR1,ModuleA_Mode1},{RR2,
       ModuleB_Mode1},{RR3,ModuleC_Mode1},{RR4,
       ModuleD_Mode1}}
10 ConfigMode,Config2,{{RR1,ModuleA_Mode2},{RR2,
       ModuleB_Mode2},{RR3,ModuleC_Mode2},{RR4,
       ModuleD_Mode2}}
11 ...
```

The words RR, RM and ConfigMode are reserved words, they are used to define a RR, RM and a configuration mode for the DRS design respectively.

The first step performed by the utility is a sanity check for the interfaces of the modes of the same RM, for DPR flow it is required to have the same number of ports for the RM modes. The sizes and names of these ports should be the same across the modes of the same RM. For Xilinx [11] tools, if this requirement is violated, the implementation of the DPR flow will fail in the place & route step, which is late in the design cycle. In our Perl utility, we do the sanity check for interfaces to catch any errors as early as possible.

The second step is to pick one configuration mode of the DRS design and generate an RTL file for this mode. Following is an example for the generated Verilog RTL file for configuration mode *Config1* in the DPR example in Fig. 1:

```
module RR1(input wire in1, in2, a_rst, clk1, output out1);
  ModuleA_mode1 ModA_1_inst (in1, in2, a_rst, clk1,out1);
endmodule
module RR2(input wire in1, in2, a_rst, clk1, output out1);
  ModuleB_mode1 ModB_1_inst(in1, in2, a_rst, clk1,out1);
endmodule
module RR3(input wire in1, in2, a_rst, clk1, output out1);
  ModuleC_mode1 ModC_1_inst(in1, in2, a_rst, clk1,out1);
endmodule
module RR4(input wire in1, in2, a_rst, clk1, output out1);
  ModuleD_mode1 ModD_1_inst(in1, in2, a_rst, clk1,out1);
endmodule
module DRS1 (input wire in1, in2, in3, in4, in5, a_rst,
     clk1, clk2, output wire out1);
  wire A_out1, B_out1, C_out1, D_out1;
  RR1 ModuleA_inst (in1, in2, a_rst, clk1, A_out1);
  RR2 ModuleB_inst (in3, A_out1, clk1, B_out1);
  RR3 ModuleC_inst (in4, B_out1, clk2, C_out1);
  RR4 ModuleD_inst (in5, C_out1, clk2, D_out1);
  assign out1 = D_out1;
endmodule
```

The third step is to generate the CDC analysis run script, the generated script is written to be run by Questa® CDC tool from Mentor Graphics to perform the CDC analysis on the design. The implemented Perl utility performs some heuristics based on the port names of the DRS design to constrain the design, as example it defines the ports match *(clk)* regular expression as clocks. Similarly, it defines the ports that match *(rst)* regular expression as resets, and define scan enable and test signals as constants. Following is an example for the generated script to run CDC analysis on configuration mode *Config1* in the DPR example in Fig. 1.

```
onerror {exit 1}
## Compile the Verilog RTL file generated from Step2
vlib work
vlog RTL_Config1.v
## Constrain the design
netlist clock clk1
netlist clock clk2
netlist reset -async -posedge a_rst
## Put the results in a separate directory
configure output directory Config1_Results
## Run CDC analysis
cdc run -d DRS1
exit 0
```

The fourth step is to run CDC analysis using Questa® CDC tool, and save the results. The Perl utility then repeats the first four steps for the all configuration modes of the design. The fifth step is to generate a report for the CDC analysis of DRS design. Following is a sample of the output report for the DRS in Fig. 1:

```
CDC Results for Mode: Config1
-----------------------------

  A) Synchronized CDC Paths:
     <None>
  B) Un-synchronized CDC Paths:
     1) From 'ModuleB_inst.ModB_1_inst.out1' (clk1)
        To 'ModuleC_inst.ModC_1_inst.out1' (clk2)
...
```

IV. Case Study

We demonstrate the value of using our CDC Verification flow by a case study of the SDR system presented in [12]. This SDR system is implemented using the DPR flow, it switches between blocks of communication standards 3G, 4G and Wi-Fi. The SDR test case has four reconfigurable modules: 1) convolutional encoder, 2) modulator, 3) Discrete Fourier Transform (DFT), and 4) Inverse Fast Fourier Transform (IFFT). Table I shows the number of modes per each block. The design has two clocks, the first clock *(clk)* is used for the channel encoder, while the other clock *(clk2)* is used for the rest of the blocks. It also has one asynchronous reset signal *(reset)*. The design has 12 operating modes. In order to perform the CDC verification using the proposed Perl utility, a CSV is provided to the utility to define the configuration modes of the design and the RTL files of the RMs as explained in Section III, following is a snippet of the created CSV file:

TABLE I
Number of modes per each RM of the design under test

Block	Number of Modes
Convolutional Encoder	4
Modulator	3
DFT	2
IFFT	3

```
1  RR,encoder
2  RR,modulator
3  RR,dft
4  RR,ifft
5  RM,conv_enc,{enc_3G_half,enc_3G_third,enc_WIFI_half,
       enc_4G_third}
6  RM,modulator,{bpsk,qpsk,qam_16}
7  RM,dft,{dft_64_point,filler_mod}
8  RM,ifft,{ifft_64,ifft_256,filler_mod}
9  ConfigMode,Config1,{{encoder,enc_3G_half},{modulator,
       bpsk},{dft,filler_mod},{ifft,filler_mod}}
10 ConfigMode,Config2,{{encoder,enc_3G_half},{modulator,
       qpsk},{dft,filler_mod},{ifft,filler_mod}}
11 ...
```

The Perl utility generates RTL design for every mode and a script to run Questa® CDC tool for CDC verification, the tool then generates a report for the CDC results for all the runs of the modes of the design.

Using our CDC verification flow, we have identified two CDC errors in all the 12 modes of the design that may cause functional errors during the operation of the system. The first error is found for the signals that are generated in clock domain of *(clk)* inside the convolutional encoder block and sampled in clock domain of *(clk2)* inside the modulator block. The modulator block design was missing synchronizing

Severity	Check	TX Signal	RX Signal	TX Clock	RX Clock
Violation (16)					
Violation	Single-bit signal does not have proper synchronizer	encoder.convo_valid_out	modulator.ff.q_tmp	clk	clk2
Violation	Multiple-bit signal across clock domain boundary	encoder.convo_out	modulator.p2s.din_s	clk	clk2
Violation	Asynchronous reset does not have proper synchronization (14)				
Violation	Asynchronous reset does not have proper synchronization	reset (14)			
Violation	Asynchronous reset does not have proper synchronization	reset	encoder.a	Async	clk
Violation	Asynchronous reset does not have proper synchronization	reset	encoder.b	Async	clk
Violation	Asynchronous reset does not have proper synchronization	reset	encoder.c	Async	clk
Violation	Asynchronous reset does not have proper synchronization	reset	encoder.convo_out	Async	clk
Violation	Asynchronous reset does not have proper synchronization	reset	encoder.convo_valid_out	Async	clk
Violation	Asynchronous reset does not have proper synchronization	reset	encoder.d	Async	clk
Violation	Asynchronous reset does not have proper synchronization	reset	encoder.e	Async	clk
Violation	Asynchronous reset does not have proper synchronization	reset	encoder.f	Async	clk
Violation	Asynchronous reset does not have proper synchronization	reset	modulator.bpsk.mod_out_im	Async	clk2
Violation	Asynchronous reset does not have proper synchronization	reset	modulator.bpsk.mod_out_re	Async	clk2
Violation	Asynchronous reset does not have proper synchronization	reset	modulator.bpsk.mod_valid_out	Async	clk2
Violation	Asynchronous reset does not have proper synchronization	reset	modulator.ff.q_tmp	Async	clk2
Violation	Asynchronous reset does not have proper synchronization	reset	modulator.p2s.din_s	Async	clk2
Violation	Asynchronous reset does not have proper synchronization	reset	modulator.p2s.ps	Async	clk2

Fig. 4. CDC results from Questa CDC tool for one of the 4G configuration modes of the design.

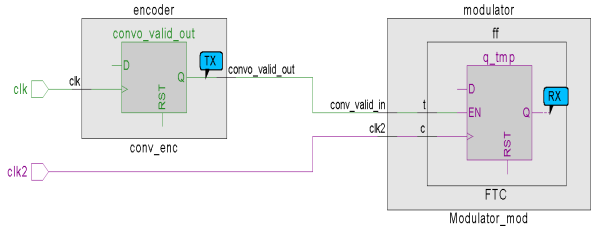

Fig. 5. Schematic of the first CDC violation in Fig. 4.

these CDC signals to clock domain of *(clk2)*, which may cause metastability issues for the registers in the modulator block.

The second error shows up due to the usage of an asynchronous reset signal *(reset)*. The asynchronous reset signal was used without being synchronized to the clock domains of *(clk)* and *(clk2)*. This may cause metastability issues for the registers in the design, because an asynchronous reset signal will be de-asserted asynchronous to the clock signal of the register, so it may violate the reset recovery time requirement for the register. Recovery time is the minimum required time to the next active clock edge after the reset is released.

The Questa® CDC results for one of the 4G modes of the design is shown in Fig. 4, the first two violations are related to the first CDC error (i.e. signals cross from encoder to the modulator), while the other 14 violations are related to the second CDC error (i.e. missing synchronization of the asynchronous reset). The schematic of the first CDC error is shown in Fig. 5. The design has to be fixed by using CDC data synchronizers for the crossing signals, and an asynchronous reset synchronizer for the *(reset)* signal. Our proposed can be used again to verify the design after the design is fixed to make sure no more issues CDC issues exist in the design.

V. Conclusion

CDC verification for digital designs is essential due to the usage of multiple clock domains in the modern designs. The CDC verification for DRS designs is a challenging task due to the lack of CAD tools support for DRS designs and the multiple operating modes of the design. In this paper, we presented a complete automated flow for CDC verification for DRS designs. Designers can use this flow with no extra effort to create new setup for CDC verification, and it can be easily integrated in the design and verification cycle of DRS designs. Using a case study from literature we demonstrated how the proposed CDC verification flow identifies couple of real CDC errors in the design which were overlooked during the design cycle.

Acknowledgment

This research was partially funded by ONE Lab at Cairo University, Zewail City of Science and Technology, and KAUST.

References

[1] A. Kamaleldin, I. Ahmed, A. M. Obeid, A. Shalash, Y. Ismail and H. Mostafa, "A Cost-Effective Dynamic Partial Reconfiguration Implementation Flow for Xilinx FPGA," in *2017 New Generation of CAS (NGCAS)*, Genova, 2017, pp. 281-284.

[2] I. Ahmed, A. Kamaleldin, H. Mostafa and A. N. Mohieldin, "Utilizing Dynamic Partial Reconfiguration to Reduce the Cost of FPGA Debugging," in *2018 16th IEEE International New Circuits and Systems Conference (NEWCAS)*, Montreal, 2018, pp. 1-4.

[3] C. Cummings, "Clock Domain Crossing (CDC) Design & Verification Techniques Using SystemVerilog," Proc. Synopsys User Group Meeting (SNUG), 2008; *http://www.sunburst-design.com/papers/CummingsSNUG2008Boston_CDC.pdf*.

[4] I. Robertson, J. Irvine, P. Lysaght, and D. Robinson,"Improved Functional Simulation of Dynamically Reconfigurable Logic," in *Field Programmable Logic and Applications (FPL), International Conference on,* 2002, pp. 541-574

[5] A. Raabe and A. Felke, "A SystemC Language Extension for High-Level Reconfiguration Modelling," in *Specification, Verification and Design Languages (FDL). Forum on,* 2008, pp. 55-60.

[6] A. Raabe, P. A. Hartmann, and J. K. Anlauf, "ReChannel: Describing and Simulating Reconfigurable Hardware in SystemC," *ACM Transactions on Design Automation of Electronic Systems (TODAES),* vol. 13, no. 1, p. 15, 2008.

[7] L. Gong and O. Diessel, "Modeling Dynamically Reconfigurable Systems for Simulation-based Functional Verification," in *Field Programmable Custom Computing Machines (FCCM), IEEE Symposium on,* 2011, pp. 9-16.

[8] L. Gong and O. Diessel, "ReSim: A Reusable Library for RTL Simulation of Dynamic Partial Reconfiguration," in *Field-Programmable Technology (FPT), International Conference on,* 2011, pp. 1-8

[9] I. Ahmed, H. Mostafa and A. N. Mohieldin, "Dynamic partial reconfiguration verification using assertion based verification," in *2018 13th International Conference on Design & Technology of Integrated Systems In Nanoscale Era (DTIS)*, Taormina, 2018, pp. 1-2.

[10] I. Ahmed, K. Nouh and A. Abbas, "Multiple reset domains verification using assertion based verification," in *2017 IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*, Abu Dhabi, 2017, pp. 1-6.

[11] Xilinx Inc. "Partial Reconfiguration User Guide UG909" v2016.1, April 2016.

[12] A. Sadek, H. Mostafa, A. Nassar and Y. Ismail, "Towards the implementation of multi-band multi-standard software-defined radio using dynamic partial reconfiguration," in *International Journal of Communications Systems,* 2017.