Journal of Circuits, Systems, and Computers
Vol. 30, No. 3 (2021) 2150042 (20 pages)

DOI: 10.1142/S0218126621500420

World Scientific
www.worldscientific.com

Functional Verification of Dynamic Partial Reconfiguration for Software-Defined Radio*

Islam Ahmed

IC Verification Solutions,
Mentor Graphics, a Siemens Business,
Cairo 11361, Egypt
islam_ahmed@mentor.com

Ahmed Nader Mohieldin[†,§] and Hassan Mostafa[†,‡,¶]

[†]*Electronics and Communications Engineering Department,*
Cairo University, Giza 12613, Egypt

[‡]*University of Science and Technology,*
Nanotechnology and Nanoelectronics Program,
Zewail City of Science and Technology,
October Gardens, 6th of October,
Giza 12578, Egypt
[§]*anader2000@yahoo.com*
[¶]*hmostafa@uwaterloo.ca*

Received 12 October 2019
Accepted 25 May 2020
Published 19 August 2020

Dynamic Partial Reconfiguration (DPR) on Field Programmable Gate Arrays (FPGAs) allows reconfiguration of some of the logic at runtime while the rest of the logic keeps operating. This feature allows the designers to build complex systems such as Software-Defined Radio (SDR) in a reasonable area. New issues can arise due to usage of DPR technique such as guaranteeing proper connections for the ports of the Reconfigurable Modules (RMs) which share the same Reconfigurable Region (RR) on the FPGA, waiting for running computations on a module before reconfiguring it, isolation of the reconfigurable modules during the reconfiguration process, and initialization of the reconfigurable module after the reconfiguration process is done. Also, the Clock Domain Crossing (CDC) verification of the dynamically reconfigurable systems 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. This paper summarizes our previous work to address these verification challenges for DPR. The approaches are demonstrated on a SDR system to show the effectiveness of applying these approaches in the design cycle.

Keywords: FPGA; software-defined; radio; partial; dynamic; reconfiguration.

*This paper was recommended by Regional Editor Piero Malcovati.
¶Corresponding author.

1. Introduction and Background

Dynamic Partial Reconfiguration (DPR) on Field Programmable Gate Arrays (FPGAs) allows reconfiguration of some of the logic at runtime while the rest of the logic keeps operating. It allows the implementation of complex circuits as Software Defined Radio (SDR) and Internet of Things (loT) applications within a reasonable area on the FPGA. Such category of designs are called Dynamically Reconfigurable Systems (DRS). Currently, Xilinx and Intel (Altera) are the main FPGA device vendors on the market. They provide a series of FPGA families that support the DPR design flow. In this paper, the Xilinx DPR design flow[1] is considered.

In DPR, the design consists of a number of Reconfigurable Modules (RMs), each module has modes that are changed during run time according to the system operating modes. A Reconfigurable Region (RR) is a location on the FPGA in which the reconfigurable module is implemented. 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 reconfigurable modules: ModuleA, ModuleB, ModuleC and ModuleD, each with four modes: Mode1, Mode2, Mode3 and Mode4. DRS designs extend the design flexibility through mapping of multiple reconfigurable modules to the same physical reconfigurable region, which reduces the design cost and the resources usage. In the example of Fig. 1, it will have four RRs on the FPGA, each RR is used for a unique RM. The RR can be configured by an RM mode according to the configuration mode of the DRS design. In the configuration mode Config1, the first RR will be loaded by the RM mode (ModuleA_Mode1), the second RR will be loaded by the RM mode (ModuleB_Mode1) and so on.

Utilizing DPR technique for FPGA designs adds a new dimension in the design and verification of FPGA designs. For Xilinx FPGAs, one of the basic requirements of a partially reconfigurable design is consistency between RMs.[1] As one module is

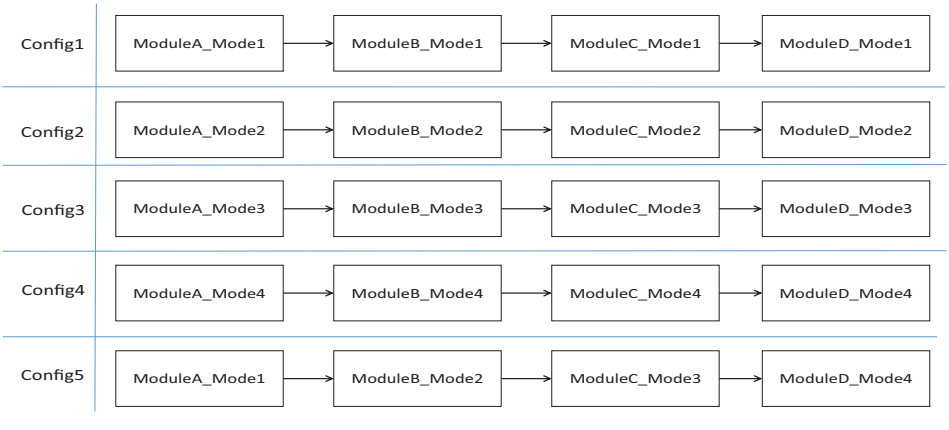

Fig. 1. An example of DPR design with three modes of configuration and one reconfigurable module per mode.

swapped for another, the connections between the static design and the RM must be identical. Such requirement adds an extra work on the designer to create a wrapper module to encapsulate all the modes of the RM, and to have a fixed interface between the static design and the RM. This interface must work for all the modes of the RM, the process of connecting the interface to different modes of the RM is an error prone task and should be verified on the Register Transfer Level (RTL) before moving to the implementation of the design on the FPGA. The detection of such connectivity issues is challenging, especially, in the early design stages of the DRS. If such errors are not tackled and verified early in the design cycle, they may cause functional errors during on-chip verification which are hard to debug.

For using DPR technique for FPGA designs, the designers must add extra logic in their DPR designs for (1) isolating the RM during the reconfiguration process, (2) initializing the RM after the reconfiguration process is done, and (3) delaying reconfiguration requests till the computations done by the RM is completed. The added logic for these tasks should be verified on the RTL to make sure it is working as expected, and any bugs are caught as early as possible in the design cycle. This paper summarizes our previous work[2] for verifying the extra added logic for DPR using Assertion-Based Verification (ABV).[3–5]

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 RTL simulated value, an example is shown in Fig. 2. Such metastability issues can cause functional errors in the design.

CDC verification[6] 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. 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 during

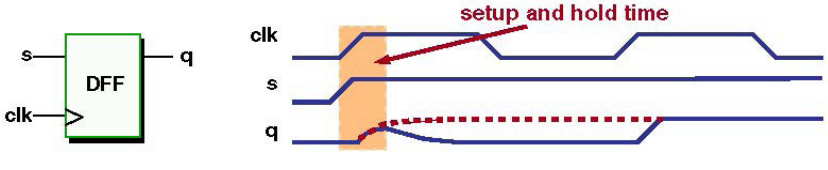

Fig. 2. Example for a metastability issue caused by CDC signal.

operation of the design or runtime verification which may cause re-spin for the design which consequently delay the delivery of the design.

This paper summarizes our research into the functional verification of the DPR technique. The addressed verification challenges are (1) guaranteeing proper connections for the ports of the Reconfigurable Modules (RMs) which share the same RR on the FPGA, (2) verification of the extra logic added in the design for DPR using ABV and (3) CDC verification for DRS. The approaches are applied on SDR design from literature to show the effectiveness of integrating these approaches in the design and verification cycle for DRS.

This paper is organized as follows. Section 2 presents the related work for verification of DPR, and Sec. 3 presents the proposed verification methodology for verification of the port connections of the RMs. Section 4 summarizes our previous work for verification of the extra added DPR. Section 5 presents the CDC verification methodology for DRS. Section 6 demonstrates a case study for applying the proposed verification methodologies. Finally, Sec. 7 draws the paper's conclusion.

2. Related Work

Several works have proposed frameworks to help in functional verification of DRS. The Dynamic Circuit Switch (DCS) method[7] adds artifacts for simulation purposes only to mimic the behavior of reconfiguration activities such as module swapping and undefined state of the RM after reconfiguration. ReChannel[8–10] is an open source SystemC library which models DPR. ReChannel adds new SystemC classes to mimic reconfiguration operations such as module swapping. The extension of ReChannel[10] proposed new classes to monitor and verify the reconfiguration details at behavioral, Transaction Level Modeling (TLM) and RTL levels. To use ReChannel, designers should be aware of using SystemC for modeling and verification of digital designs, and extra efforts are needed to setup the simulation environment on the behavioral level, TLM level, and RTL.

ReSim[11] is system verilog library built on the Open Verification Methodology (OVM) which uses a 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 immediately before, during and after the reconfiguration. So, it can detect functional bugs that were missed by DCS, ReChannel and OSSS + R. Setting up the design to use the ReSim setup needs extra effort by the designer. The ReSim library is extended[12] to support state saving and restoration of the RMs.

The existing works in literature have the following disadvantages and limitations:

- They model the DPR activities using simulation-only artifacts (i.e., non-synthesizable models), so they can't be used with formal verification methods. The

testbenches used for testing the design should thoroughly cover all the corner cases which is impractical in some cases.
- Extra effort is needed to setup the verification environment as SystemC modeling or OVM environment setup.
- When an error is caught, extra effort is needed to debug the error and pinpoint the root cause of the issue, it can be related to non-DPR logic.
- 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.

A SystemC-based design methodology (OSSS + R)[13] was proposed to automate the modeling, synthesis, and simulation of DRS designs. It automatically generates synthesizable code for the reconfiguration controller to manage the module swapping of RMs. But, it uses only pre-defined reconfiguration control mechanism, so it cannot handle all styles of DPR designs.

3. Verification of Ports Connections of Reconfigurable Modules for Dynamic Partial Reconfiguration

The first step for creating a DRS is to identify the static logic (i.e., logic that is always active in all the operating modes of the design) and the reconfigurable logic (i.e., the logic that can change from one operating mode to another) in the design. For Xilinx DPR flow,[1] the interface of a reconfigurable module should be consistent across all its modes, such requirements add an extra step in the design flow to create an RTL wrapper for each RM to encapsulate all its modes. Issues appear in this step when there is a mismatch in the number of ports between different modes of the RM, the RTL wrapper of that RM will have number of ports equal to the maximum number of ports in all the modes of the RM, in that case, the designer should take care when connecting the ports for each mode of the RM to not affect the functionality of the circuit.

An example for the design modifications needed for adopting the DPR technique is shown in Fig. 3. In this example, there is a design with two operating modes, the design has two modules: (1) *shift_reg* module which is a static module (i.e., existing in all the operating modes of the design) and (2) accum module which is a reconfigurable module and has two modes (accum1 and accum2). The number of ports of the modes of the reconfigurable module (accum) is different. The example as well shows the modified RTL after creating the RTL wrapper for the reconfigurable module, if the port *in* 3 is used in the first mode of the RR '*RR_mode1.v*', then the design functionality will be altered. Such modifications in the RTL should be verified before moving to implementing the design on the FPGA. The modification for the interfaces of the RMs is an error prone task especially for large designs which have a large number of ports for the modes of the RMs and mismatch in the sizes of these

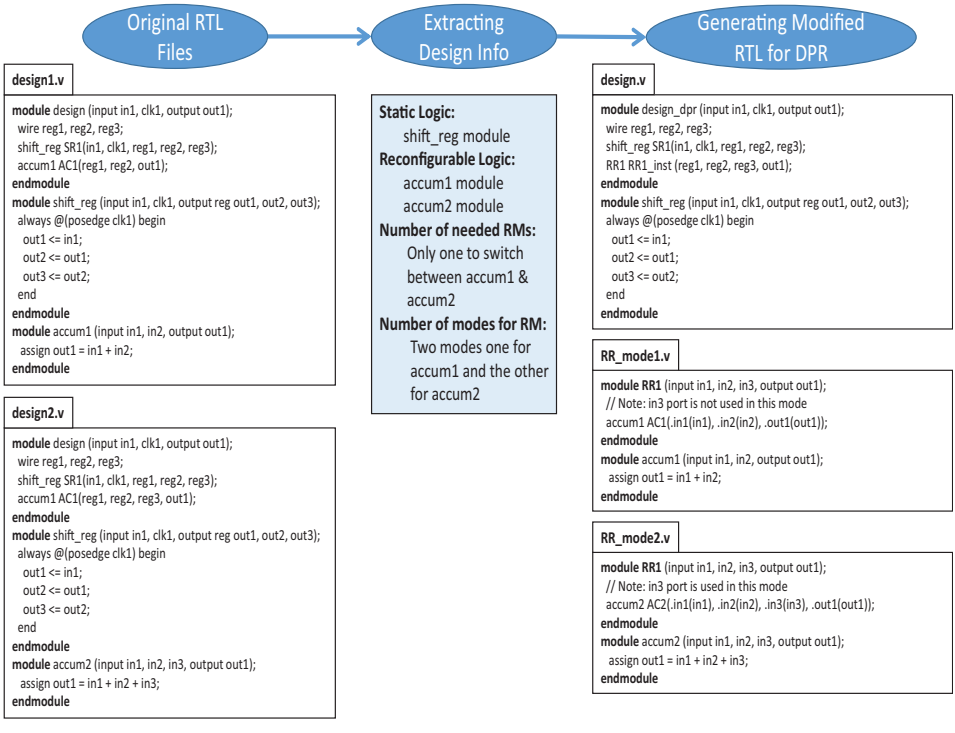

Fig. 3. Design modifications in the RTL files for DPR.

modules like the SDR case. In this paper, the connectivity verification approach[18,19] is used to verify the changes in the interfaces of the reconfigurable modules.

The verification flow is shown in Fig. 4. After the RTL files are compiled and the design is synthesized, the netlist of the design is traversed using netlist access Application Programming Interfaces (APIs) to extract the connections of the reconfigurable modules from the original design, the output of this step is a Comma Separated Values (CSV) file that lists the hierarchical paths of the RM ports and their connections. In this paper, the netlist access APIs provided by Questa tools[14] are used. Our SVA property generator takes the CSV file and write an assertion for every source and destination pair. The following SVA property is generated for every source and destination pair to verify their connection:

```
property connect_pair (clock, source, destination);
  @(posedge clock) disable iff(~(`RM_MODE_ENABLE))
     (source==destination);
endproperty
```

where *RM_MODE_ENABLE* is a macro that can be set by the designer such that when its logic value is 1, it indicates a specific RM mode is active. This macro is different from one RM mode to another because only one RM mode can be active at a

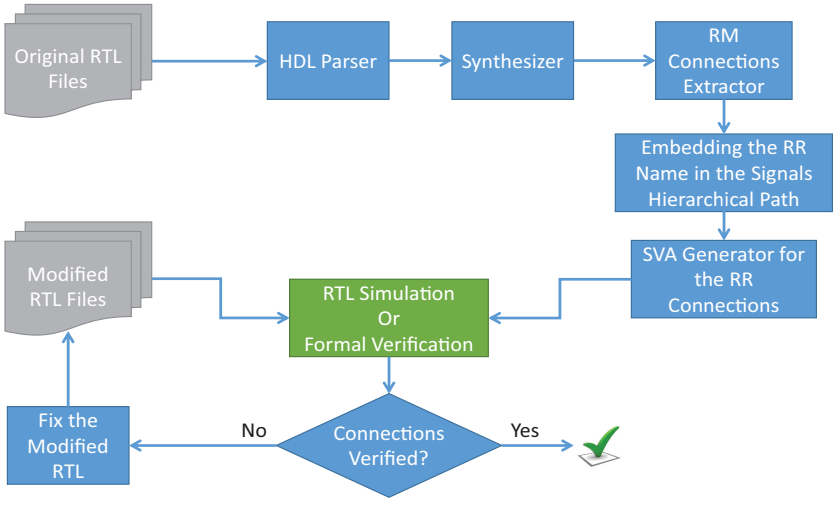

Fig. 4. Verification flow to verify the connections of the RMs.

time. For each RM mode, there will be a separate CSV file to test its connections, and consequently unique set of assertions. The assertions generated for each mode can be verified using RTL simulation or formal verification.

4. Verification of Dynamic Partial Reconfiguration Logic Using Assertion-Based Verification

The typical structure of designs that utilize DPR is shown in Fig. 5, the Internal Configuration Access Port (ICAP)[1] is used to read or write to the FPGA

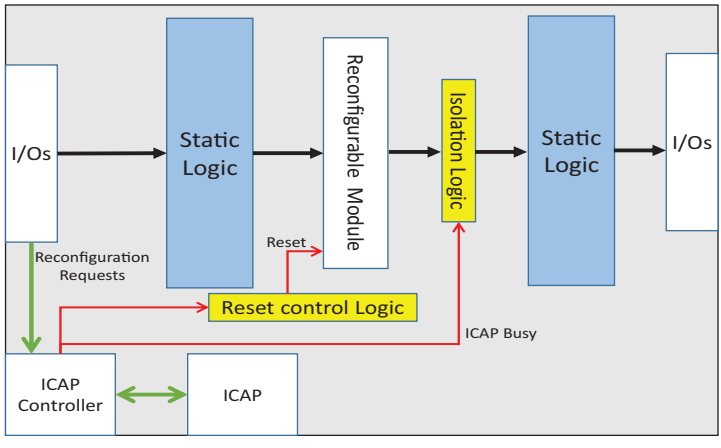

Fig. 5. Typical structure of a design that utilizes DPR.

configuration memory. A controller is needed for the ICAP to handle the reconfiguration requests and monitor its status. The output port of the ICAP can be used to monitor its status. The yellow blocks in Fig. 5 are added by the designer to have a correct operation for the design during and after the reconfiguration process.[1] Our verification approach is to model the functionality of the DPR logic using SVA[15] properties, then verify these properties on the design using formal or simulation methods.

4.1. *Isolation logic*

During the reconfiguration process of the RM, the values of newly downloaded bitstream may drive incorrect values to the static logic side, so designers add isolation logic for all the ports of the RM to prevent the propagation of the data from the RM to the static logic during the reconfiguration process. The typical structure of designs that utilize DPR is shown in Fig. 5, the ICAP is used to read or write to the FPGA configuration memory. A controller is needed for the ICAP to handle the reconfiguration requests, to handle the control of the ICAP and monitor its status. The output port of the ICAP can be used to monitor its status. The yellow blocks in Fig. 5 are added by the designer to have a correct operation for the design during and after the reconfiguration process. For the isolation logic, it is verified using the following SVA property for every output port of the RM:

```
property verify_isol(clock, source, destination, ICAP_BUSY);
 @(posedge clock)
 (($changed(source) && ICAP_BUSY) |=> $stable(destination));
endproperty
```

where the *source* signal is an output port of the RM, the *destination* signal is the register driven by the output port on the static side, and the *ICAP_BUSY* is the signal which indicates that there is a reconfiguration process in progress.

4.2. *Reset control logic for the RM*

After the reconfiguration process is done, the sequential elements of the RM should be reset to guarantee proper operation of the circuit. If the RM is not reset after reconfiguration, the state of sequential elements will be undefined and may be affected by erroneous values from the previous RMs that share the same physical area on the FPGA. The reset control logic is verified using the following SVA property:

```
property verify_reset(clock, RM_reset, ICAP_BUSY);
 @(posedge clock)
 ($fall(ICAP_BUSY) |-> $rose(RM_reset));
endproperty
```

where *RM_reset* is the reset signal of the RM, and the *ICAP_BUSY* is the signal which indicates that there is a reconfiguration process in progress. The assertion

property implies that when the *ICAP_BUSY* is changed from logic value 1 to 0 (i.e., the reconfiguration through ICAP is done), then the reset signal of the RM should be asserted to reset all the sequential elements of the RM.

4.3. *Synchronizing the reconfiguration process*

When a computation is being done in the RM, the designers want to block any reconfiguration request until such computation is done. Such mechanism is required in applications like SDR, when a packet is being processed for WiFi standard as example, it should be processed completely before switching to any other standard like 3G or 4G. The synchronization of the reconfiguration requests is verified using the following SVA property:

```
property verify_sync(clock, RM_busy, ICAP_GO);
 @(posedge clock)
 ($rose(ICAP_GO) until $fall(RM_busy));
endproperty
```

where *RM_busy* is the signal which indicates that a computation is being done by the RM, and *ICAP_GO* is the control signal which tells the ICAP to start a new reconfiguration process. For some applications, it is not needed to check such synchronization, as it is acceptable to flush the data of the RM.

5. Proposed CDC Verification Flow for DRS Designs

The proposed CDC Verification flow is shown in Fig. 6. A Perl utility is implemented to automate the flow. The inputs for the utility are as follows: (1) RTL files of RMs modes, (2) RTL wrapper for DRS design and (3) 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 provided by the designer, so there is no extra effort needed for creation of these files to use the proposed CDC verification flow. The 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
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
```

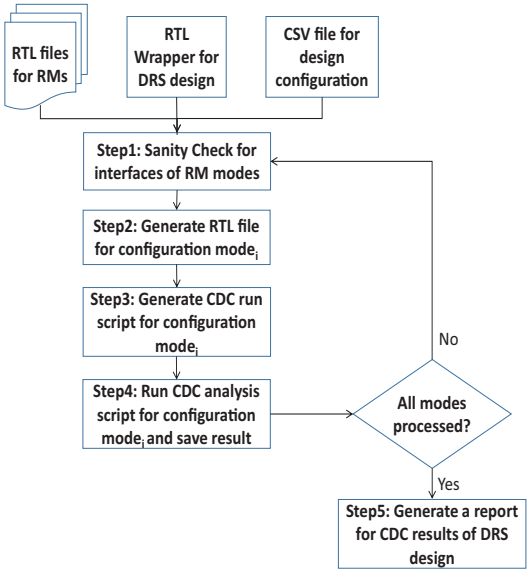

Fig. 6. Proposed CDC verification flow.

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
// In each operating mode:
//      A mode for ModuleA will be instantiated here
endmodule
module RR2(input wire in1, in2, a_rst, clk1, output out1);
// Empty
// In each operating mode:
//      A mode for ModuleB will be instantiated here
endmodule
module RR3(input wire in1, in2, a_rst, clk1, output out1);
// Empty
// In each operating mode:
//      A mode for ModuleC will be instantiated here
endmodule
module RR4(input wire in1, in2, a_rst, clk1, output out1);
// Empty
// In each operating mode:
//      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 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 following CSV file is an example for the DPR design, in Fig. 1:

```
RR,RR1
RR,RR2
RR,RR3
RR,RR4
RM,ModuleA,{ModuleA_Mode1,ModuleA_Mod2,ModuleA_Mode3,ModuleA_Mode4}
RM,ModuleB,{ModuleB_Mode1,ModuleB_Mod2,ModuleB_Mode3,ModuleB_Mode4}
RM,ModuleC,{ModuleC_Mode1,ModuleC_Mod2,ModuleC_Mode3,ModuleC_Mode4}
RM,ModuleD,{ModuleD_Mode1,ModuleD_Mod2,ModuleD_Mode3,ModuleD_Mode4}
ConfigMode,Config1,{{RR1,ModuleA_Mode1},{RR2,ModuleB_Mode1},{RR3,
    ModuleC_Mode1},{RR4,ModuleD_Mode1}}
ConfigMode,Config2,{{RR1,ModuleA_Mode2},{RR2,ModuleB_Mode2},{RR3,
    ModuleC_Mode2},{RR4,ModuleD_Mode2}}
ConfigMode,Config3,{{RR1,ModuleA_Mode3},{RR2,ModuleB_Mode3},{RR3,
    ModuleC_Mode3},{RR4,ModuleD_Mode2}}
ConfigMode,Config4,{{RR1,ModuleA_Mode4},{RR2,ModuleB_Mode4},{RR3,
    ModuleC_Mode4},{RR4,ModuleD_Mode4}}
ConfigMode,Config5,{{RR1,ModuleA_Mode1},{RR2,ModuleB_Mode2},{RR3,
    ModuleC_Mode3},{RR4,ModuleD_Mode4}}
```

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[1] tools, if this requirement is violated, the implementation of the DPR flow will fail in the place and route step, which is late in the design cycle. In our Perl utility, the sanity check for interfaces is done 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. The 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 defines scan enable and test signals as constants. The 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 all the configuration modes of the design. The fifth step is to generate a report for the CDC analysis of DRS design. The 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)
...
```

6. Case Study

The verification approaches presented in this paper for verification of DPR is applied on an SDR chain.[16,17] 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 1 shows the number of modes per

Table 1. Number of modes per each RM of the design under test.

Block	Number of modes
Convolutional encoder	4
Modulator	3
DFT	2
IFFT	3

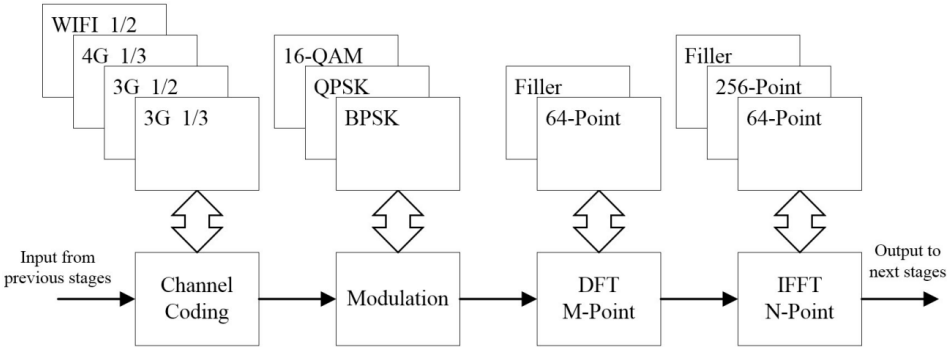

Fig. 7. Block diagram of the design under test.[16]

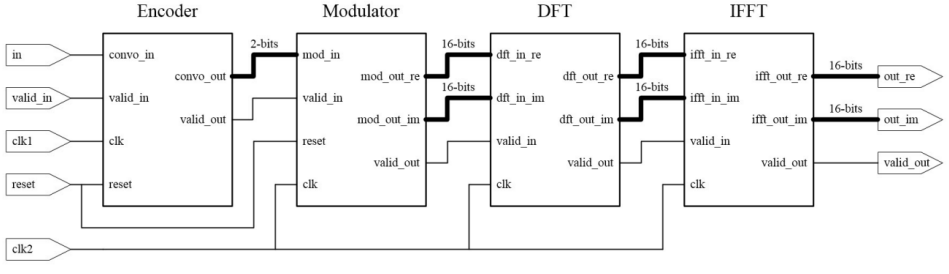

Fig. 8. Schematic of the design under test.[16]

each block. The block diagram and the schematic of the designs are shown in Figs. 7 and 8, respectively.

6.1. *Verification of ports connections of the RMs and the DPR logic*

The port connections are verified using SVA properties as explained in Sec. 3. The port connections should be verified for every mode of each RM, the number of assertions generated for verifying port connections is proportional to the number of

source	destination
LTE_top.convo_in	LTE_top.xlxi_1.convo_in
LTE_top.clk	LTE_top.xlxi_1.clk
LTE_top.reset	LTE_top.xlxi_1.convo_reset
LTE_top.convo_valid_in	LTE_top.xlxi_1.convo_valid_in
LTE_top.xlxi_1.convo_out	LTE_top.xlxn_18
LTE_top.xlxi_1.convo_valid_out	LTE_top.xlxn_17

Fig. 9. CSV file extracted for connections of the first mode of the convolutional encoder block.

```
connect_pair_0: assert property(connect_pair(LTE_top.clk, LTE_top.convo_in,LTE_top.xlxi_1.convo_in));
connect_pair_1: assert property(connect_pair(LTE_top.clk, LTE_top.clk,LTE_top.xlxi_1.clk));
connect_pair_2: assert property(connect_pair(LTE_top.clk, LTE_top.reset, LTE_top.xlxi_1.convo_reset));
connect_pair_3: assert property(connect_pair(LTE_top.clk, LTE_top.convo_valid_in,LTE_top.xlxi_1.convo_valid_in));
connect_pair_4: assert property(connect_pair(LTE_top.clk, LTE_top.xlxi_1.convo_out,LTE_top.xlxn_18));
connect_pair_5: assert property(connect_pair(LTE_top.clk, LTE_top.xlxi_1.convo_valid_out, LTE_top.xlxn_17));
```

Fig. 10. Assertions generated for connections of the first mode of the convolutional encoder block.

	Name	Time
	checkers_inst.connect_pair_0	2s
	checkers_inst.connect_pair_1	2s
	checkers_inst.connect_pair_2	2s
	checkers_inst.connect_pair_3	2s
	checkers_inst.connect_pair_4	2s
	checkers_inst.connect_pair_5	2s

Fig. 11. Results of Questa Formal tool for the assertions generated for connections of the first mode of the convolutional encode block, all assertions are proven.

modes and the number of ports for each mode. The SVA properties are run on the DPR design using Questa Formal tool.[14] Figure 9 shows an example for the CSV file extracted for the first mode of the convolutional encoder RM. Figure 10 shows example for the generated assertions to verify the port connections in the CSV file of Fig. 9, and Fig. 11 shows the results of Questa Formal tool in which all the assertion properties are proven.

Table 2 shows the number of ports for every RM, and Table 3 shows the number of assertions generated for verification of port connections, isolation logic, reset control logic, and the synchronization logic.

The number of assertions for the isolation logic equals to the number of output ports for all the RMs, the number of assertions for the reset control logic equals to the number of RMs as each RM will have its own reset control logic, and only one assertion is generated to test the synchronization logic of the DPR controller.

Table 2. Ports information for RMs of the design under test.

Block	Total no. of ports	No. of output ports
Convolutional encoder	6	2
Modulator	7	3
DFT	7	3
IFFT	7	3

Table 3. Generated assertions for DPR verification.

Goal	Number of assertions
Port connections	6*4 + 7*3 + 7*2 + 7*3 = 80
Isolation logic for output ports	11
Reset control logic	1*4
Synchronization logic	1
Total	96

All the assertion properties of the port connections are proven by the Questa Formal tool. But, the Questa PropCheck tool reports firings when applying the assertions for isolation logic, reset control logic and synchronization logic, the following three bugs were identified in the design under test:

(1) The output ports of the RMs were not isolated during the reconfiguration process. This should be fixed in the design such that the output ports of the RMs are totally isolated from the static logic during the reconfiguration process to avoid the propagation of any erroneous values from the RMs to the static logic.
(2) The reset signals of the RMs were not activated right after the completion of the reconfiguration process. This should be fixed in the design such that the reset signals should be asserted after the reconfiguration to put the RM in a defined initial state before operation.
(3) The DPR controller was not handling the case in which a new reconfiguration request is received when the RM is still processing data.

6.2. *CDC verification of the DRS*

In this section, the value of using our CDC verification flow is demonstrated. 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, to perform the CDC verification using the proposed Perl utility, the following CSV is provided to the utility for the configuration

modes of the design with the RTL files of the reconfigurable modules as explained in Sec. 3:

```
RR,encoder
RR,modulator
RR,dft
RR,ifft
RM,conv_enc,{enc_3G_half,enc_3G_third,enc_WIFI_half,enc_4G_third}
RM,modulator,{bpsk,qpsk,qam_16}
RM,dft,{dft_64_point,filler_mod}
RM,ifft,{ifft_64,ifft_256,filler_mod}
ConfigMode,Config1,{{encoder,enc_3G_half},{modulator,bpsk},{dft,filler_mod},{ifft,filler_mod}}
ConfigMode,Config2,{{encoder,enc_3G_half},{modulator,qpsk},{dft,filler_mod},{ifft,filler_mod}}
ConfigMode,Config3,{{encoder,enc_3G_half},{modulator,qam_16},{dft,filler_mod},{ifft,filler_mod}}
ConfigMode,Config4,{{encoder,enc_3G_third},{modulator,bpsk},{dft,filler_mod},{ifft,filler_mod}}
ConfigMode,Config5,{{encoder,enc_3G_third},{modulator,qpsk},{dft,filler_mod},{ifft,filler_mod}}
ConfigMode,Config6,{{encoder,enc_3G_third},{modulator,qam_16},{dft,filler_mod},{ifft,filler_mod}}
ConfigMode,Config7,{{encoder,enc_WIFI_half},{modulator,bpsk},{dft,filler_mod},{ifft,filler_mod}}
ConfigMode,Config8,{{encoder,enc_WIFI_half},{modulator,qpsk},{dft,filler_mod},{ifft,filler_mod}}
ConfigMode,Config9,{{encoder,enc_WIFI_half},{modulator,qam_16},{dft,filler_mod},{ifft,filler_mod}}
ConfigMode,Config10,{{encoder,enc_4G_third},{modulator,bpsk},{dft,dft_64},{ifft,ifft_256}}
ConfigMode,Config11,{{encoder,enc_4G_third},{modulator,qpsk},{dft,dft_64},{ifft,ifft_256}}
ConfigMode,Config12,{{encoder,enc_4G_third},{modulator,qam_16},{dft,dft_64},{ifft,ifft_256}}
```

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, two CDC errors have been identified 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 designs were missing synchronizing 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

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

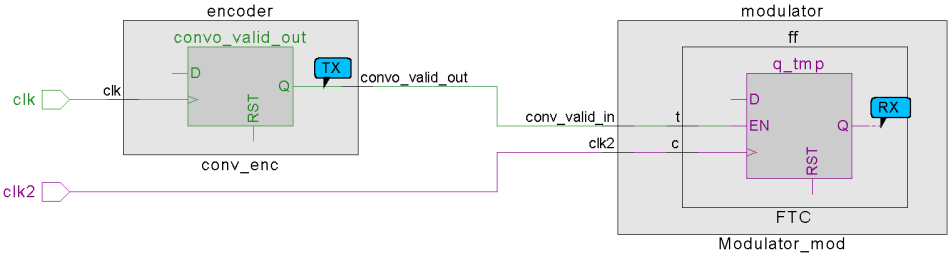

Fig. 13. Schematic of the first CDC violation in Fig. 12.

4G modes of the design is shown in Fig. 12, 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. 13. 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 method can be used again to verify the design after the design is fixed to make sure no more CDC issues exist in the design.

7. Conclusion

The DPR technique has been used recently to implement complex systems such as SDR systems. Using the DPR technique in digital designs needs more verification efforts to make sure the design functionality is not altered after adopting the DPR technique. In this paper, three functional verification approaches are presented for DPR to verify: (1) the port connections of the RMs, (2) the dedicated logic added for DPR activities, and (3) CDC signals in the designs. The port connections of the RMs and the DPR dedicated logic are verified using ABV, the assertion properties can be verified in simulation or formal verification. For CDC verification, it is a challenging task for DRS designs due to the lack of CAD tools that support this kind of designs.

In this paper, a complete automated flow for CDC verification is presented for DRS designs. Designers can use this flow with no extra effort to create a new setup for CDC verification, and it can be easily integrated into the design and verification cycle of DRS designs. The verification approaches should be done on the RTL design before moving to implement it on the FPGA to make sure of the correct functionality of the design, as any error caught during verification will force the designs to restart the implementation cycle after fixing the functional errors in the design. Using a case study from literature, the paper demonstrated how the proposed verification flows identify functional issues.

In Sec. 2, the previous works which addressed the verification of DPR were presented. The previous works have the following disadvantages and limitations:

(1) They model the DPR activities using simulation-only artifacts (i.e., non-synthesizable models), so they cannot be used with formal verification methods. The testbenches used for testing the design should thoroughly cover all the corner cases which is impractical in some cases.
(2) Extra effort is needed to set up the verification environment such as SystemC modeling or OVM environment setup.
(3) When an error is caught, extra effort is needed to debug the error and pinpoint the root cause of the issue, it can be related to non-DPR logic.
(4) They focused 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.

This paper has proposed new methodologies for functional verification of DPR. The proposed methodologies in this paper have some advantages when compared to the existing works in literature:

(1) It models the connections of the RMs using SVA properties and verifies them using ABV. The assertions can be used for formal verification or RTL simulation. Also, it can be integrated with any previous work that performs RTL simulation.
(2) It enhances the observability, reduces the debug time, and improves error detection. When an SVA assertion property fails in RTL simulation or formal verification, it pinpoints to the root cause of the issue with no extra effort.
(3) The assertions can be synthesized on the FPGA to perform runtime verification for DPR, this is not covered in this paper.
(4) It automates a framework for the CDC verification for designs which use DPR.

Acknowledgment

This paper was funded by Mentor Graphics and ONE Lab at Zewail City of Science and Technology/Cairo University.

References

1. Xilinx, Partial Reconfiguration User Guide UG909 (2016).
2. I. Ahmed, H. Mostafa and A. Mohieldin, Dynamic partial reconfiguration verification using assertion based verification, *13th IEEE Int. Conf. Design & Technology of Integrated Systems in Nanoscale Era (DTIS)* (2018), pp. 1–2.
3. M. Litterick, Assertion-based verification using system verilog, 2007, http://www.verilab.com/files/svug_2007_abv_litterick.pdf.
4. I. Ahmed, H. Mostafa and A. Mohieldin, On the functional verification of dynamic partial reconfiguration, *IEEE 61th Int. Midwest Symp. Circuits and Systems (MWSCAS)* (2018), pp. 1–4.
5. I. Ahmed, H. Mostafa and A. Mohieldin, Automatic clock domain crossing verification flow for dynamic partial reconfiguration, *IEEE 61th Int. Midwest Symp. Circuits and Systems (MWSCAS)* (2018), pp. 1–4.
6. 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.
7. I. Robertson, J. Irvine, P. Lysaght and D. Robinson, Improved functional simulation of dynamically reconfigurable logic, *Int. Conf. Field Programmable Logic and Applications (FPL)* (2002), pp. 541–574.
8. A. Raabe and A. Felke, A systemC language extension for high-level reconfiguration modelling, *Forum on Specification, Verification and Design Languages (FDL)* (2008), pp. 55–60.
9. A. Raabe, P. A. Hartmann and J. K. Anlauf, ReChannel: Describing and simulating reconfigurable hardware in SystemC, *ACM Trans. Des. Autom. Electron. Syst.* **13** (2008) 1–8.
10. L. Gong and O. Diessel, Modeling dynamically reconfigurable systems for simulation-based functional verification, *IEEE Symp. Field Programmable Custom Computing Machines (FCCM)* (2011), pp. 9–16.
11. L. Gong and O. Diessel, ReSim: A reusable library for RTL simulation of dynamic partial reconfiguration, *Int. Conf. Field-Programmable Technology (FPT)* (2011), pp. 1–8.
12. L. Gong and O. Diessel, Functionally verifying state saving and restoration in dynamically reconfigurable systems, *ACM/SIGDA Int. Symp. Field Programmable Gate Arrays* (2012), pp. 241–244.
13. A. Schallenberg, W. Nebel, A. Herrholz and P. A. Hartmann, OSSS + R: A framework for application level modelling and synthesis of reconfigurable systems, *Design, Automation and Test in Europe (DATE)* (2009), pp. 970–975.
14. Mentor Graphics, Questa CDC and formal functional verification, 2017, www.mentor.com/products/fv/questa-formal-verification-apps.
15. IEEE Standard for SystemVerilog — Unified Hardware Design, Specification, and Verification Language.
16. A. Sadek, H. Mostafa and A. Nassar, On the use of dynamic partial reconfiguration for multiBand/multiStandard software defined radio, *Int. Conf. Electronics, Circuits, and Systems (ICECS)* (2015), pp. 498–499.
17. A. Sadek, H. Mostafa, A. Nassar and Y. Ismail, Towards the implementation of multi-band multi-standard software-defined radio using dynamic partial reconfiguration, *Int. J. Commun. Syst.* **30** (2017) 1–10.

18. H. Saafan, M. Watheq and A. Salem, SoC connectivity specification extraction using incomplete RTL design: An approach for formal connectivity verification, *Int. Design & Test Symp. (IDT)* (2016), pp. 110–114.
19. Mentor Graphics, Questa connectivity check formal application, 2019, www.mentor.com/products/fv/questaconnectivity-check.