Partial Reconfiguration on ZedBoard

Efe ARIN

Gazi University Department of Electrical-Electronic Engineering
efearin@gazi.edu.tr

Abstract

This report is prepared to give some brief information about project 'simple partial reconfiguration setup on ZedBoard evaluation board', done as part of ELE519 course in TOBB ETU.

Keywords: Partial Reconfiguration, PR, PR on ZedBoard

1. Introduction

At this report, firstly, brief information will be given about what partial reconfiguration is and its scope of usage, then structure of partial reconfiguration will be mentioned along with some remarkable points. After that implementation codes will be explained and implementation process will be detailed.

Vivado version 2015.4 is used and Vivado version related documentations of Xilinx ug909 Partial Reconfiguration (v2016.1)[1] and ug947 Partial Reconfiguration Tutorial (v2016.1)[2] documents along with ZedBoard Manual[3] are referenced.

2. What is PR

PR (partial reconfiguration) is a tool on FPGAs', allows dynamic change in active design without any disturbance on unrelated running parts. For example if there are two noncontemporary process named as A1 and A2. PR allows the system to switch between those process without affecting the rest of the implementation using ICAPs' (Internal Configuration Access Port) of SLRs' (Super Logic Region).

To do that, FPGA design is divided into two different type of logic as static and reconfigurable. Static design is structured as top level with black boxes for reconfigurable modules. Full bit file with one of the configurable modules, say A1 and A2 in this case, and partial bit files for each of the reconfigurable modules are created. When system is started to process with one of the partial modules then any of the partial bit file is fed to system to change inner structure of the black boxes of the static design over ICAP, which results change in functionality (Figure 2.1).

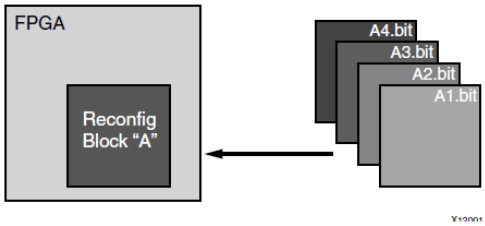

Figure 2.1: Basic PR concept [2]

3. Why PR

Although flat design is better both in implementation simplicity and speed -since optimization across reconfigurable boundaries are prohibited, exclusive placement and complex routing containment are needed and there is a need for extra timing slack for reconfiguration- in some cases, basically lack of resources and power consumption issues, PR is the solution. Some benefits of PR in that way are listed below as;

• Reducing the size of the FPGA required to implement a given function, with consequent reductions in cost and power consumption
• Providing flexibility in the choices of algorithms or protocols available to an application
• Enabling new techniques in design security
• Improving FPGA fault tolerance
• Accelerating configurable computing

By those listed above, PR could be efficiently used in network multiport interfaces to switch among communication protocols, dynamically reconfigurable pocket processors to change processing functions and asymmetric key encryptions for key managements.

4. Considerations

Some important points should be considered are listed below.

- There is no project support for PR. Only way is to use Tcl.
- Floorplanning is required to define reconfigurable regions in hardware. Vertical alignment to frame/clock region boundaries increases efficiency on 7 series devices (so on ZedBoard too) and enables RESET_AFTER_CONFIG property.
- A PR design must consider the initiation of Partial Reconfiguration as well as the delivery of partial BIT files, either within the FPGA or as part of the system design.
- A Reconfigurable Partition must contain a super set of all pins to be used by the varying Reconfigurable Modules implemented for the partition. If an RM (Reconfigurable Module) uses different inputs or outputs from another RM, the resulting RM inputs or outputs might not connect inside of the RM. The tools handle this situation for all unused inputs and outputs. The output is tied to a constant value and the value of the constant can be controlled by HD.PARTPIN_TIEOFF property on the unused output pin.
- For user reset signals, determining if the logic inside the RM is level or edge sensitive is important. If the reset circuit is edge sensitive (as it may be in some IP such as FIFOs), then the RM reset should not applied until after configuration is complete.
- Hierarchical design techniques should be followed for better performance
- Routing resources and components like CLB, BRAM, and DSP are reconfigurable; however, clocks and clock modifying logics including components BUFG, BUFR, MMCM and PLL are not.
- I/O and I/O related components (ISERDES, OSERDES, IDELAYCTRL), Serial transceivers (MGTs) and related components and individual architecture feature components (such as BSCAN, STARTUP, ICAP, XADC.) should be placed in static part only.
- Before PR implementation non-PR setup should be checked to prevent further implementation problems will occur in PR.
- If PR partition needs too many connections to connect static part as a superset of PR modules related, routing congestion can occur.

5. Implementation

For implementation Xilinx PR tutorial is referenced and projected to ZedBoard. Manipulated files, that will be explaned, could be found on ……..

General route is as follows;

1. Synthesize both static and reconfigurable modules independently
2. Create PBlocks for configurable partitions on hardware
3. Assign configurable Pblocks as reconfigurable from their properties
4. Implement full design (static with all reconfigurable partitions are filled)
5. Checkpoint save for fully routed design
6. Remove all reconfigurable modules and checkpoint save
7. Lock routing and static placement
8. Assign reconfigurable partitions with other reconfigurable modules and repeat up to 8 until no reconfigurable module checkpointing left
9. Verify configurations
10. Create bitstreams and program device

In this example, there are four reconfigurable module hdl codes as count_down, count_up, shift_left, shift_right and static module as top could be found under folder ../sources/hdl. Top module has two black box for counter and shifter. In that example count_up, shift_right and count_down, shift_left are combined such as one configuration count up and shift to right at the same time and remaining two for other one (Figure 5.1).

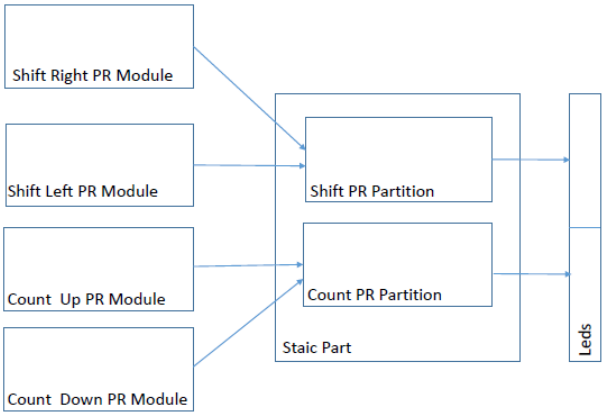

Figure 5.1: Example Flow

5.1 Synthesis

By running design.tcl at the root folder from Tcl Shell using command

```
source design.tcl -notrace
```

synthesis is done. File design.tcl contains parameter declaration and Tcl commands needed just for PR synthesis process. In the file, first, Tcl substructures -like design_utility, hd_floorplan_utility- that will be used during synthesis process are sourced. Those Tcl resources could be found under folder ../Tcl with their related command explanations at file README.txt in the same folder. Then device (ZedBoard) is specified with its device code and package name using `check_part`. After that flow control is planned to do just synthesis (to add additional process set related values to 1) and output and input directories are specified. Then a static module with

name attribute as top is created by using top.v file under input folder ../Sources/hdl/top and reconfigurable modules are created in the same way. Lastly implementations -one of them contains count up and shift right and other one contains count down and shift left at the reconfigurable partitions- are created. All done is passed to run.tcl under ../Tcl folder that does required process for the specified flow controls and outputs modules related synthesis checkpoints to ../Synth folder and log files as run.log, command.log and critical.log to the root folder.

5.2 Assembling and Floorplanning

If the file, created at previous step, top_synth.dcp at ../Synth/Static/ is opened using IDE. It could be seen from netlist pane that there are black boxes (Figure 5.2.1) as inst_shift and inst_count using;

```
read_checkpoint -cell inst_shift
Synth/shift_right/shift_synth.dcp
```

and

```
read_checkpoint -cell inst_count
Synth/count_up/count_synth.dcp
```

these black boxes are filled by shift right and count up accordingly (Figure 5.2.1).

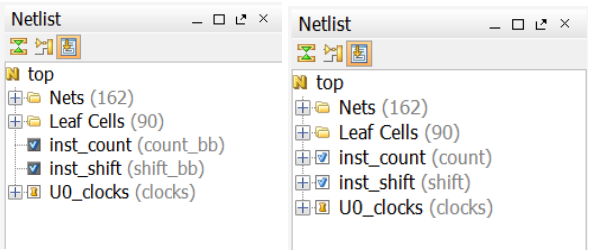

Figure 5.2.1: Black boxes as inst_count and inst_shift on the left and attributed modules on the right

Using codes, that manipulate HD.RECONFIGURABLE property;

```
set_property HD.RECONFIGURABLE 1
[get_cells inst_shift]

set_property HD.RECONFIGURABLE 1
[get_cells inst_count]
```

these submodules defined as partial reconfigurable parts and this created design checkpoint could be saved using;

```
write_checkpoint
./Checkpoint/top_link_right_up.dcp
```

After that there is a need for floor planning. Places that reconfigurable modules are placed on ZedBoard, should be specified. For each configurable partition PBlocks could be drawn. To do that reconfigurable submodule, that will be placed on, should be right clicked and Floorplaning>Draw PBlock should be chosen. After that, on device view, PBlock could be placed. In this case submodules are placed at different clock regions. Clock and other inputs to reconfigurable modules can be decoupled to prevent spurious writes to memories during reconfiguration, to prevent such problems RESET_AFTER_RECONFIG should be enabled at regions. Moreover make sure that resource requirements of modules are satisfied and interconnect boundaries shouldn't be split. In case of resource problems SNAPPING_MODE could be activated from the Properties tab of PBlocks or error guidance could be used by getting DRC report from Tools>Report>Report DRC (doing DRC report for just Partial Reconfiguration is enough). After that floorplan (Figure 5.2.1) could be saved using

```
write_xdc ./Sources/xdc/fplan.xdc
```

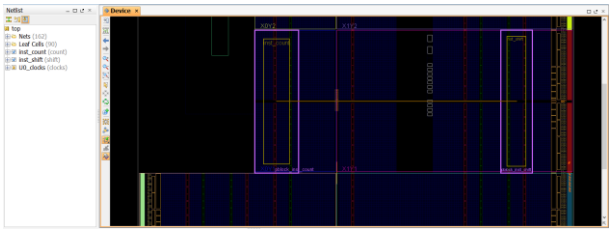

Figure 5.2.1: Floorplanned design

5.3 Implementation

By using

```
read_xdc Sources/xdc/top_io.xdc
```

constraints are gathered. For more detail, see file itself. Optimization, placing and routing the design could be done using codes

```
opt_design

place_design

route_design
```

in order. With placing, physical interface points between static and reconfigurable parts appear on device view as white boxes. Those pins could be found under cell pins tab in the cell properties pane. Routing uses those pins. In the routed design using routed design view all nets could be seen. Full design checkpoint and related reports could be saved using

```
write_checkpoint -force
Implement/Config_shift_right_count_up_imp
lement/top_route_design.dcp

report_utilization -file
Implement/Config_shift_right_count_up_imp
lement/top_utilization.rpt
```

```
report_timing_summary -file
Implement/Config_shift_right_count_up_imp
lement/top_timing_summary.rpt
```

optionally reconfigurable partition check points could be saved using

```
write_checkpoint -force -cell inst_shift
Checkpoint/shift_right_route_design.dcp

write_checkpoint -force -cell inst_count
Checkpoint/count_up_route_design.dcp
```

For the other part of reconfiguration reconfigurable modules should be cleaned to fed by new ones using codes below

```
update_design -cell inst_shift -black_box

update_design -cell inst_count -black_box
```

Before fed configurable parts with new modules all placement and routing should be locked using

```
lock_design -level routing
```

Now all routed nets on device view seen as dashed lines instead of straight and placed components orange instead of blue, which shows they all locked (Figure 5.3.1).

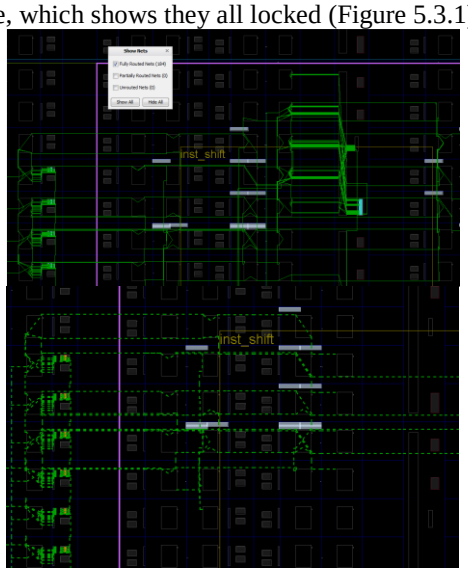

Figure 5.3.1: Unlocked (up) and locked (bottom) design [2]

Static part check point could be get using

```
write_checkpoint -force
Checkpoint/static_route_design.dcp
```

For the other checkpoint of partial reconfigurable parts (shift left and count down) modules should be set again then optimized, placed and routed and saved. Now implementation files could be found under implement folder. Using them bitstreams could be generated.

5.4 Generating Bitstreams

First open the file of checkpoint that bitstream files will be generated using

```
open_checkpoint
Implement/Config_shift_right_count_up_imp
lement/top_route_design.dcp
```

Then create bitstreams using

```
write_bitstream -force -file
Bitstreams/Config_RightUp.bit
```

Now 3 bit files are created under folder ../Bitstreams one for full design and 2 partial ones. Do the same for left-down design. Moreover, using bitstream generated for just static part with black boxes may be used to keep system at the state idle (when there is no need for reconfigurable module) which decreases the power consumption. To do that open the static design check point using

```
open_checkpoint
Checkpoint/static_route_design.dcp
```

Do all reconfigurable partition pins as buffer using

```
update_design -cell inst_count -
buffer_ports

update_design -cell inst_shift -
buffer_ports
```

Then place and route design using codes used previously and get a bitstream from it using

```
write_bitstream -force -file
Bitstreams/config_black_box.bit
```

Now all bitstream files are in folder ../Bitstreams (Figure 5.4.1).

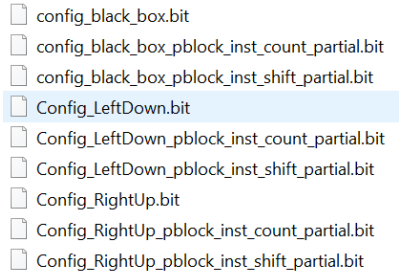

Figure 5.4.1: All created bitstream files

5.5 Configuring FPGA

To see results, open Vivado and click on Flow>Open Hardware Manager and connect to the ZedBoard. Then click Program Device and program it with Config_ReightUp.bit under folder ../Bitstreams. See the leds on board, half of them counts up binary while the other half shifting right. Then program it with Config_LeftDown_pblock_inst_shift_partial.bit and see led shift direction is changed while counting up is not affected. By programing the device with other partial and full bitfiles changes might be observed. Moreover the button assigned could be used to halt the process.

6. References

[1] Anon, (2016). Vivado Design Suite User Guide: Partial Reconfiguration. [online] Available at: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2015_1/ug909-vivado-partial-reconfiguration.pdf [Accessed 13 Aug. 2017].

[2] Anon, (2016). Vivado Design Suite Tutorial: Partial Reconfiguration. [online] Available at: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2016_1/ug947-vivado-partial-reconfiguration-tutorial.pdf [Accessed 13 Aug. 2017].

[3] Anon, (2014). ZedBoard. [online] Available at: http://ZedBoard.org/sites/default/files/documentations/ZedBoard_HW_UG_v2_2.pdf [Accessed 13 Aug. 2017].