

DPRC and Partial Reconfiguration Design Flow – Quick Start Guide

The aim of this document is to provide a quick overview and reference guide on designing a LEON3-based dynamically reconfigurable system on Xilinx FPGAs. This guide shows how to instantiate and use the Dynamic Partial Reconfiguration Controller (DPRC) with DMA AHB interface to perform FPGA self-dynamic partial reconfiguration through the Internal Configuration Access Port (ICAP).

The partial reconfiguration design flow is also detailed taking into account an example application, represented by a system where designer wants to instantiate an adaptive FIR filter peripheral.

The filter is mapped on the AHB and APB, and it is able to adapt its internal structure depending on several run-time parameters (e.g., timing budget, power, required Msamples-per-second, data type, fault-tolerance, set of coefficients). To achieve filter adaptation the designer has two alternatives:

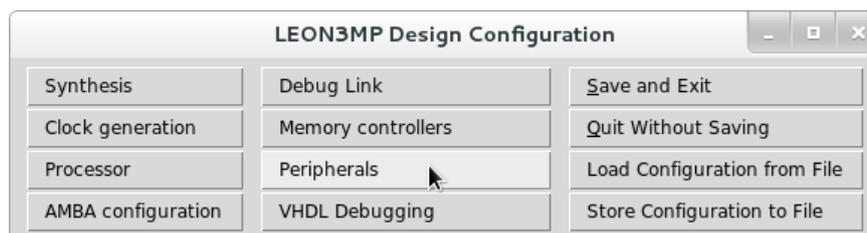
1. design a complex adaptive FIR filter, including all the aforementioned characteristics that can be enabled by setting at run-time some configuration registers;
2. design several non-adaptive FIR filters, each one including a subset of characteristics, covering all the possible real use cases.

The first solution is not efficient since, in most cases, not all the functionalities are enabled at the same time, sometimes making a relevant part of the hardware resources useless. Instead, the second solution requires less area, since using FPGA dynamic reconfiguration hardware resources can be time-multiplexed among different implementations of the same FIR filter. Moreover, if the target application makes the filter inactive for long periods, it can completely erase the configuration memory area associated to the peripheral, thus saving power, or just time-multiplex the same FPGA region with other signal processing cores.

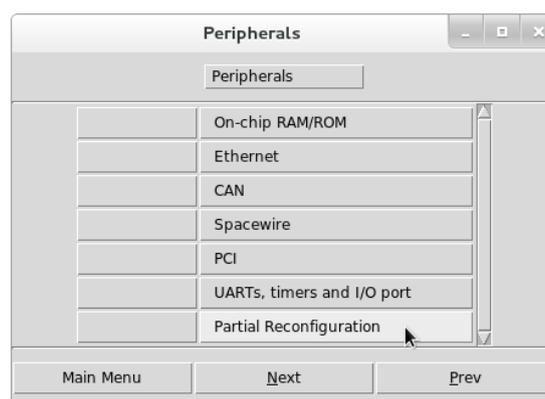
1 Instantiating DPRC in a LEON3-based system

To enable self-dynamic partial reconfiguration through the ICAP port, DPRC must be instantiated and connected on both AHB and APB.

If the target is a reference template design included in GRLIB, issuing *make xconfig* allows instantiating DPRC through a graphical interface. The *Partial Reconfiguration* sub-menu can be found under *Peripherals* section of the LEON3MP Design Configuration GUI (Figure 1.a-b).



(a)



(b)

Figure 1. How to reach Partial Reconfiguration setup after issuing *make xconfig*.

According to Figures 2 – 5, the user can enable/disable the instantiation of DPRC in the target template design, and select one of the three operating modes (i.e., *Synchronous*, *Asynchronous* or *D²PRC*).

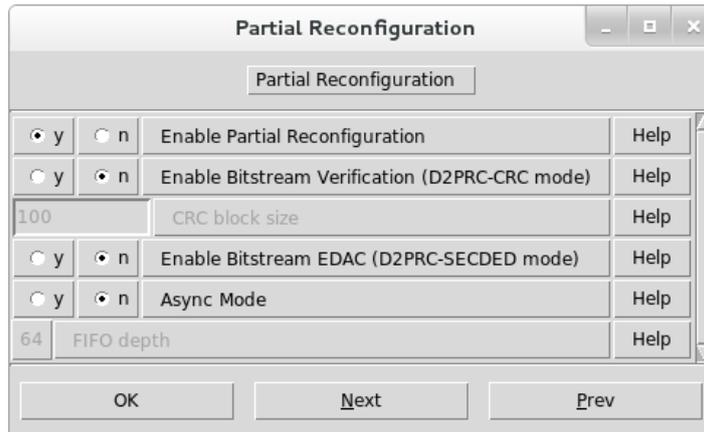


Figure 2. DPRC configured in Synchronous mode.

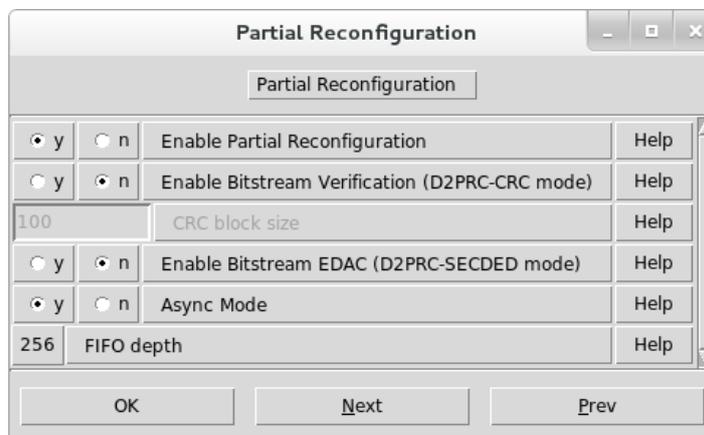


Figure 3. DPRC configured in Asynchronous mode.

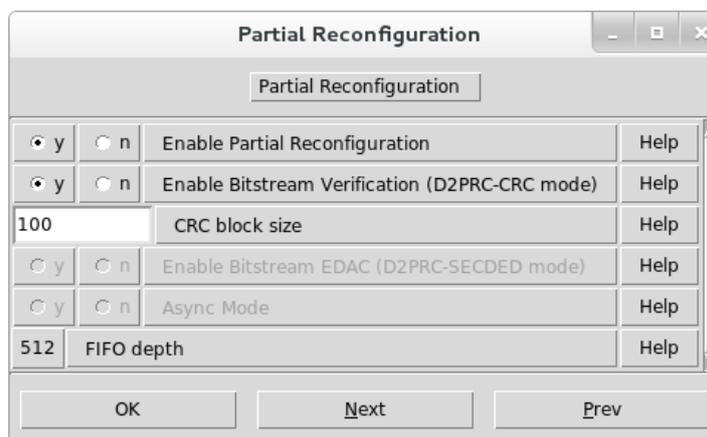


Figure 4. DPRC configured in Dependable CRC (D²PRC-CRC) mode.

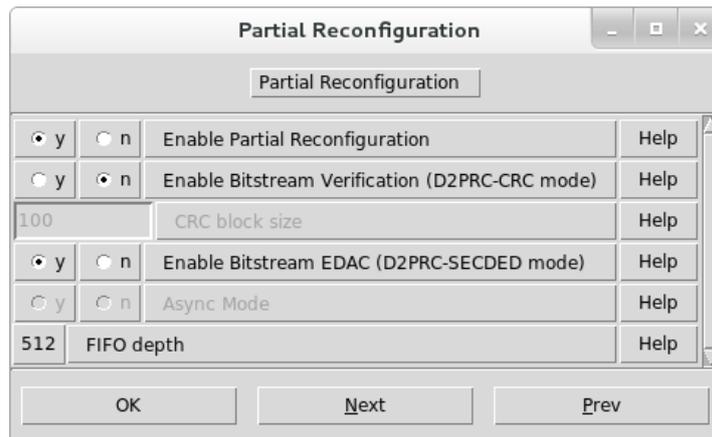


Figure 5. DPRC configured in *Dependable EDAC (D²PRC-SECEDED)* mode.

When configured in *D²PRC-CRC* mode, the user can set how many 32-bit bitstream words must be grouped in a CRC-block. Whenever a CRC-block is completely loaded from the memory, a CRC check is performed to verify bitstream integrity before delivering data to ICAP.

In addition, if *Asynchronous* or *D²PRC* modes are selected, user can choose the depth of the internal FIFO buffer, in terms of 32-bit words.

Alternatively, designer can directly instantiate DPRC in the HDL description of the system as shown in following example:

```

library ieee;
use ieee.std_logic_1164.all;
library grlib;
use grlib.amba.all;
use grlib.devices.all;
library techmap;
use techmap.gencomp.all;
library testgrouppolito;
use testgrouppolito.dprc_pkg.all;

entity dprc_ex is
    port (
        clk : in std_ulogic;
        rstn : in std_ulogic;

        ..... -- other signals
    );
end;
architecture rtl of dprc_ex is

    -- AMBA signals declarations
    . . .

begin

    -- AMBA Components are instantiated here
    . . .

```

```

-- Dynamic Partial Reconfiguration Controller

p1 : dprc
    generic map(hindex => 2, pindex => 11, paddr => 11, pirq => 11, technology => virtex4,
crc_en => 1, edac_en => 0, words_block => 50, fifo_dcm_inst => 0, fifo_depth => 9, cfg_clkmul => 2,
cfg_clkdiv => 1, raw_freq => 50000, clk_sel => 0)
    port map( rst => rstn, clk => clk, ahbmi => ahbmi, ahbmo => ahbmo(2), apbi => apbi, apbo =>
apbo(11), rm_reset => rm_reset_sig, clkraw => clk_board, clk100 => '0');

end;

```

This instantiation sets DPRC as Master 2 on the AHB, and Slave 11 on the APB. Moreover, D^2PRC mode is selected ($crc_en='1'$, $edac_en='0'$) with CRC block size equal to 50 words. A 512 32-bit words FIFO depth has been chosen ($fifo_depth=9$) and the 100 MHz clock for the ICAP domain is generated internally ($clk_sel='0'$) by the $clkgen$ component included in DPRC. This clock is synthesized starting from the clk_board signal (connected to DPRC input port $clkraw$). Its frequency is 50 MHz ($raw_freq=50000$), therefore cfg_clkmul and cfg_clkdiv are set in order to double the frequency.

2 Partial Reconfiguration Design Flow (ISE Design Suite 14.7)

This section describes how to build a partially reconfigurable system using ISE Design Suite 14.7 exploiting the Partial Reconfiguration design flow. This flow can be used when dealing with Vitex-4/5/6 FPGAs. A very similar approach is also used in Vivado for 7 Series FPGAs and will be discussed in the next section.

2.1 Design Requirements

The inputs for creating a partially reconfigurable design are represented by netlists, HDL files and constraints. Some design requirements must be fulfilled to not incur in implementation errors.

A hierarchical design methodology is required for the modules that must be reconfigured. First, the designer must be aware of which logic must reside in the so called “static” part of the design (i.e., not reconfigurable) and which one is allowed to be reconfigured at run-time. Taking as example the aforementioned FIR filter peripheral with APB/AHB interfaces, one can try to minimize the amount of logic that must be reconfigured between two versions of the filter (to limit reconfiguration overheads) by splitting the bus interface from the actual processing core. Consequently, if the APB/AHB interface will be the same among different filter implementations, it can reside on the static part of the design. Therefore, the reconfigurable logic will be composed of the actual filter processing core, only. This situation is illustrated in *Figure 6*, where DPRC and FIR modules have been instantiated in a LEON3-based system. The AHB/APB interface of the filter belongs to the static part of the design, while different filter core implementations, also called reconfigurable modules (i.e., *FIR V1*, *FIR V2*, *FIR Vn*), can be swapped in order to time-multiplex the same hardware resources.

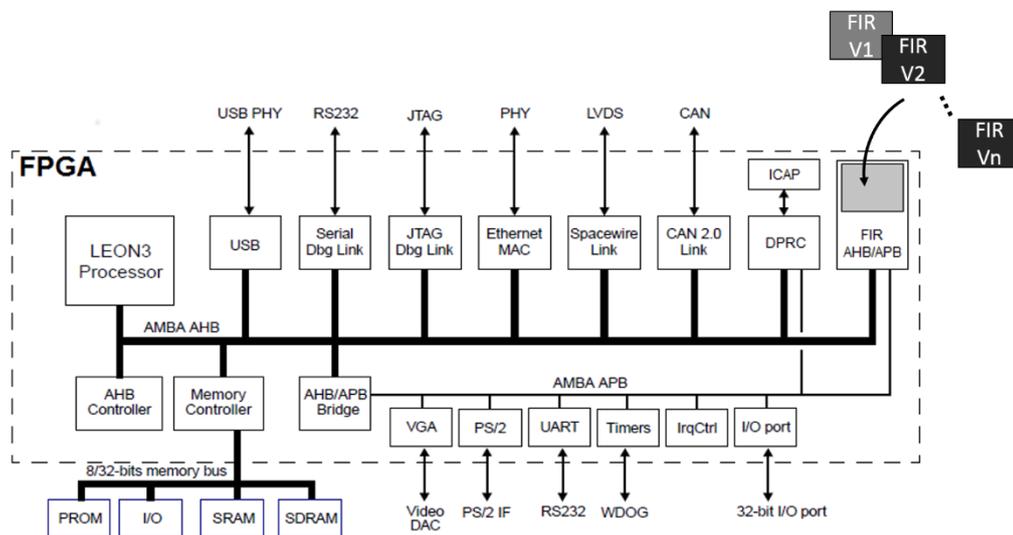


Figure 6. LEON3-based system including DPRC and FIR AHB/APB

The reconfigurable logic will be confined in a reconfigurable partition of the design. A reconfigurable partition must contain a super set of all I/Os to be used by the varying reconfigurable modules implemented in that partition. Probably, this will lead to unused inputs or outputs for some reconfigurable module versions. The unused inputs will be left dangling inside of the module and will cause the implementation tool to issue messages that the designer may ignore. In the case of a reconfigurable module where there are unused output ports, these outputs can be tied off to a logical constant. In other words, each reconfigurable module version must show the same entity interface to the upper level (i.e., the entity in which it is instantiated). Then, in its internal architecture some unused outputs must be tied to a constant value, and some inputs can be left unused.

Reconfigurable modules cannot include some types of components. In particular, BUFG, BUFR, MMCM, PLL, DCM, I/O and I/O related components, Serial transceivers (MGTs) and related components, individual architecture feature components (such as BSCAN, STARTUP, XADC, etc.) must remain in the static region of the design. In addition, bidirectional interfaces between static and reconfigurable regions are not allowed.

During Partial Reconfiguration, because the reconfigurable logic is modified while the FPGA device is operating, the static logic connected to outputs of reconfigurable modules must ignore data from them. The reconfigurable modules will not output valid data until Partial Reconfiguration is complete and the reconfigured logic is reset. Therefore, some decoupling logic must be inserted between static and reconfigurable regions. A common design practice is to register all output signals (on the static side of the interface) from the reconfigurable module. An enable signal can be used to isolate the logic until it is completely reconfigured. Moreover, reconfigurable modules must be locally reset to ensure a known starting condition after reconfiguration. These last two requirements can be satisfied exploiting the *rm_reset* DPRC output, which is asserted during a reconfiguration process, and can be used as synchronous reset and/or enable signal for decoupling logic and reconfigurable modules.

In the chosen example, the least significant bit of *rm_reset* output signal has been connected to the *rm_reset* input port of the FIR peripheral and it is used internally as reset for the reconfigurable modules and enable signal for the decoupling logic.

```

library ieee;
use ieee.std_logic_1164.all;
library gplib;
use gplib.amba.all;
use gplib.devices.all;
library techmap;
use techmap.gencomp.all;
library testgrouppolito;
use testgrouppolito.dprc_pkg.all;

entity leon3mp is
    port (
        clk : in std_ulogic;
        rstn : in std_ulogic;

        ..... -- other signals
    );
end;
architecture rtl of leon3mp is

    -- AMBA signals declarations
    . . .

    signal rm_reset_sig : std_logic_vector(31 downto 0);
begin

    -- AMBA Components are instantiated here
    . . .

    -- Dynamic Partial Reconfiguration Controller
    p1 : dprc
        generic map(hindex => 2, pindex => 11, paddr => 11, pirq => 11, technology => virtex4,
        crc_en => 1, edac_en => 0, words_block => 50, fifo_dcm_inst => 0, fifo_depth => 9, cfg_clkmul => 2,
        cfg_clkdiv => 1, raw_freq => 50000, clk_sel => 0)
        port map( rst => rstn, clk => clk, ahbmi => ahbmi, ahbmo => ahbmo(2), apbi => apbi, apbo =>
        apbo(11), rm_reset => rm_reset_sig, clkraw => clk_board, clk100 => '0');

    -- FIR filter
    dir_ex : fir_ahb_dma_apb
        generic map(hindex => 3, pindex => 12, paddr => 12, technology => virtex4, pmask => 16#fff#)
        port map( rstn => rstn, clk => clk, ahbmi => ahbmi, ahbmo => ahbmo(3), apbi => apbi, apbo =>
        apbo(12), rm_reset => rm_reset_sig(0));

end;

```

2.2 Netlists generation

Once DPRC and FIR modules are instantiated in the system, the synthesis project can be created (issuing *make scripts* if targeting a GRLIB template). In order to synthesize the static portion of the design, the design files related to reconfigurable modules must be removed from the synthesis project. *Figure 7* shows the hierarchy of the design, where the sources associated to the *fir_core* instance have been removed, while its associated interfaces (e.g., *DMA2AHB* and *syncram* components) have been left in the project.

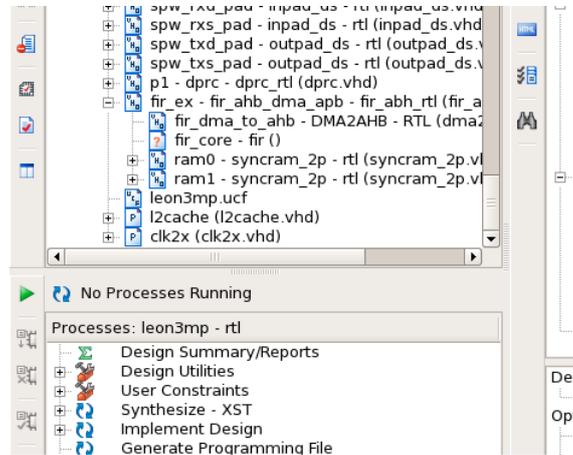


Figure 7. Design hierarchy with the removed *fir_core* instance

The output of the synthesis (.ngc file) represents the netlist of the static portion of the design.

Reconfigurable modules sources must be synthesized independently from the rest of the design. For each reconfigurable module a netlist is required. Moreover, it is important to disable I/O buffers insertion (*-iobuf* switch for XST) since, as aforementioned, I/Os and I/Os related components cannot reside in reconfigurable regions (*Figure 8*).

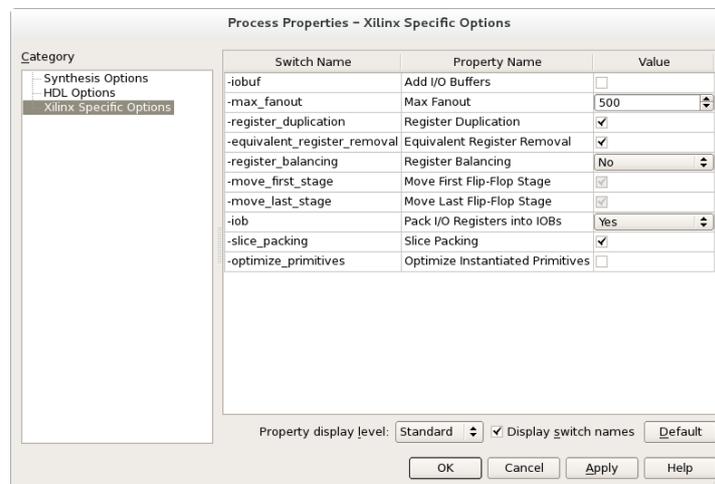


Figure 8. Add I/O Buffers disabled before running synthesis

2.3 Implementation

After running synthesis for both static and reconfigurable modules, the Xilinx PlanAhead tool can be launched to setup a partial reconfiguration project. The project can be created from the graphical user interface or using a tcl script.

When creating a new project in PlanAhead, select *Post-synthesis project*, since netlists are already available, and check *Enable Partial Reconfiguration* switch (*Figure 9*).

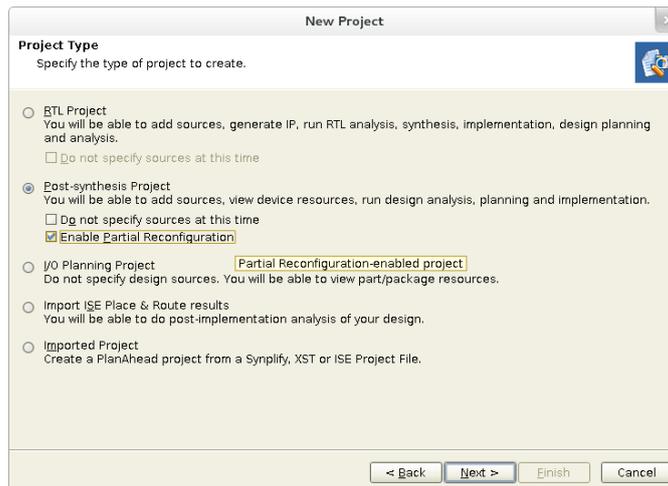


Figure 9. Create a Partial Reconfiguration project using PlanAhead

User is then asked to include netlists and constraints associated to the static part of the design. If targeting Virtex-4 or Virtex-5 devices, constraints must be also applied to the ICAP I/O signals, as suggested in the Xilinx Partial Reconfiguration User Guide. In fact, ICAP primitive is not considered a synchronous element and paths connected to it are not included in the timing analysis. A reference constraint file (*icapv4v5.ucf*) is included in $\langle \text{GRLIB root} \rangle / \text{lib/testgroup/polito/pr/}$.

After opening the synthesized design, a warning message is prompted since tool recognizes that some modules are missing from the imported static netlist. In fact, reconfigurable modules have not been yet included in the design (Figure 10).

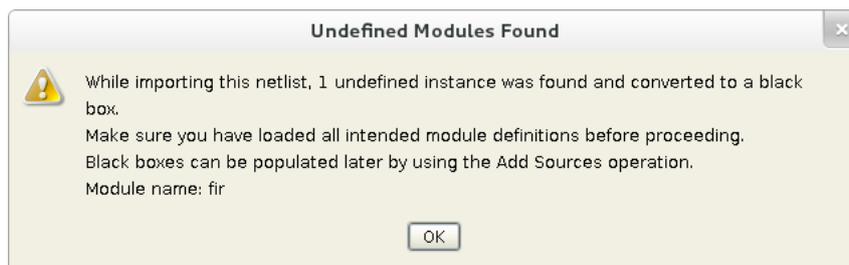


Figure 10. Warning message displayed after importing "static" netlists

Module names reported in the warning message must correspond to the entity names of the missing reconfigurable modules.

Then, a reconfigurable partition for the reconfigurable modules can be defined. This partition will be associated to all the possible implementations of the FIR filter core. After importing the "static" netlists and constraints, the netlist hierarchy is shown (Figure 11). Right-click on the instance associated to the reconfigurable modules, select *Set Partition*, and then choose "is a Reconfigurable Partition". The *Add Reconfigurable Module* window will be displayed (Figure 12). Netlists and constraints associated to the first reconfigurable module (i.e., first version of the FIR filter) can be loaded selecting "Netlist already available for this Reconfigurable module". To load additional reconfigurable modules, right-click on the associated instance in the netlist hierarchy, and click on *Add Reconfigurable Module*.

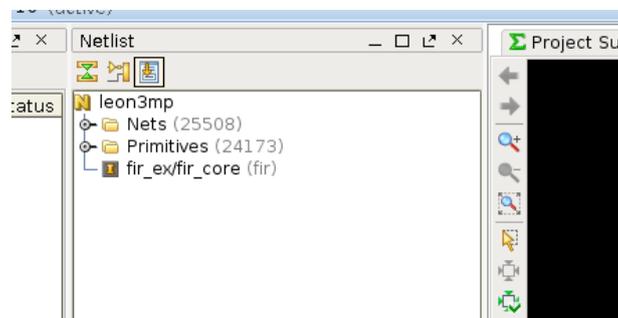


Figure 11. Netlist hierarchy. Highlighted the undefined instance of the reconfigurable modules

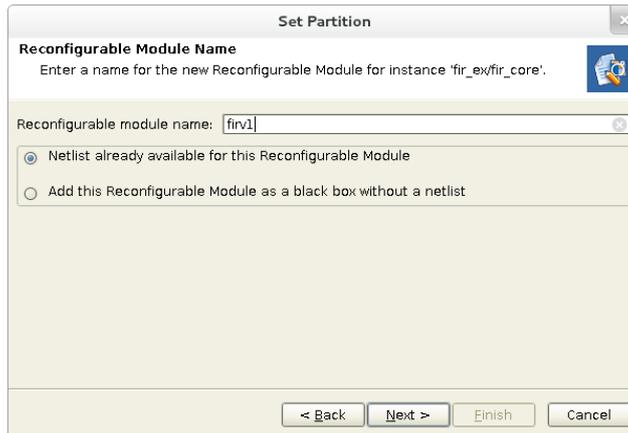


Figure 12. Add Reconfigurable Module window

Eventually, a *Black-box*, i.e., an empty, reconfigurable module can be also included in the reconfigurable partition by selecting “*Add this Reconfigurable Module as a black box without a netlist*” in the Add Reconfigurable Module window (Figure 13).

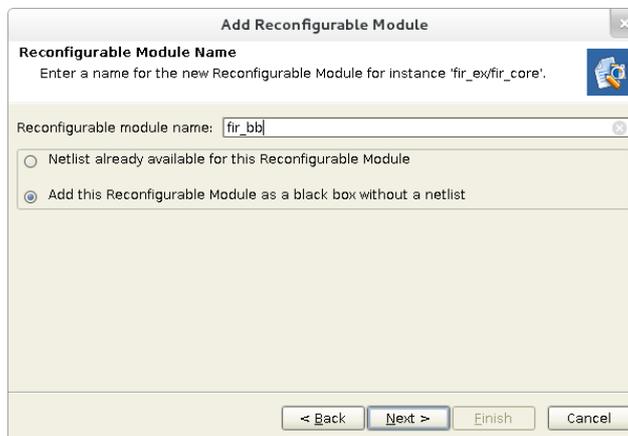


Figure 13. Add a Black-box reconfigurable module

The design hierarchy should list the imported reconfigurable modules for the selected partition, as shown in Figure 14.

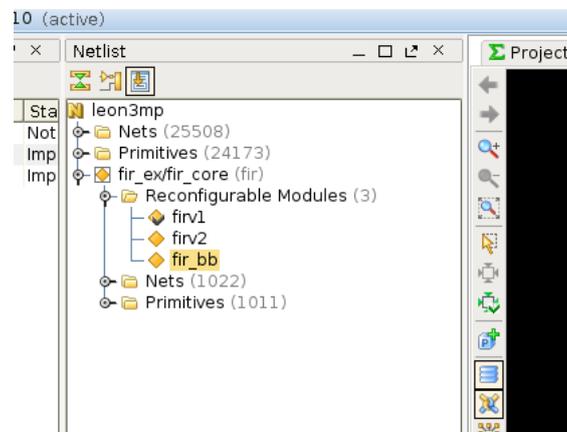


Figure 14. Design hierarchy showing three reconfigurable modules for the *fir_ex/fir_core* partition

In particular, the netlist hierarchy in *Figure 14* shows that the instance *fir_ex/fir_core* includes three reconfigurable modules, i.e., it can be implemented in three different ways (*firv1*, *firv2* or *firbb*).

Afterwards, physical constraints for the reconfigurable partition must be defined. Right-click on the reconfigurable partition instance and select **Set Pblock Size**. The device view will appear and a rectangle can be drawn in order to manually define the physical region of the FPGA in which the reconfigurable partition will be implemented. The area must be sized in order to accommodate the reconfigurable module that requires more resources with respect to the other versions. The required (estimated) resources for a particular module version are shown in the **Pblock Properties** tab. As example, *firv1* requires 12 DSP48 slices. This means that the area drawn on the device window must include at least 12 DSPs, otherwise an error will be asserted during implementation (*Figure 15*).

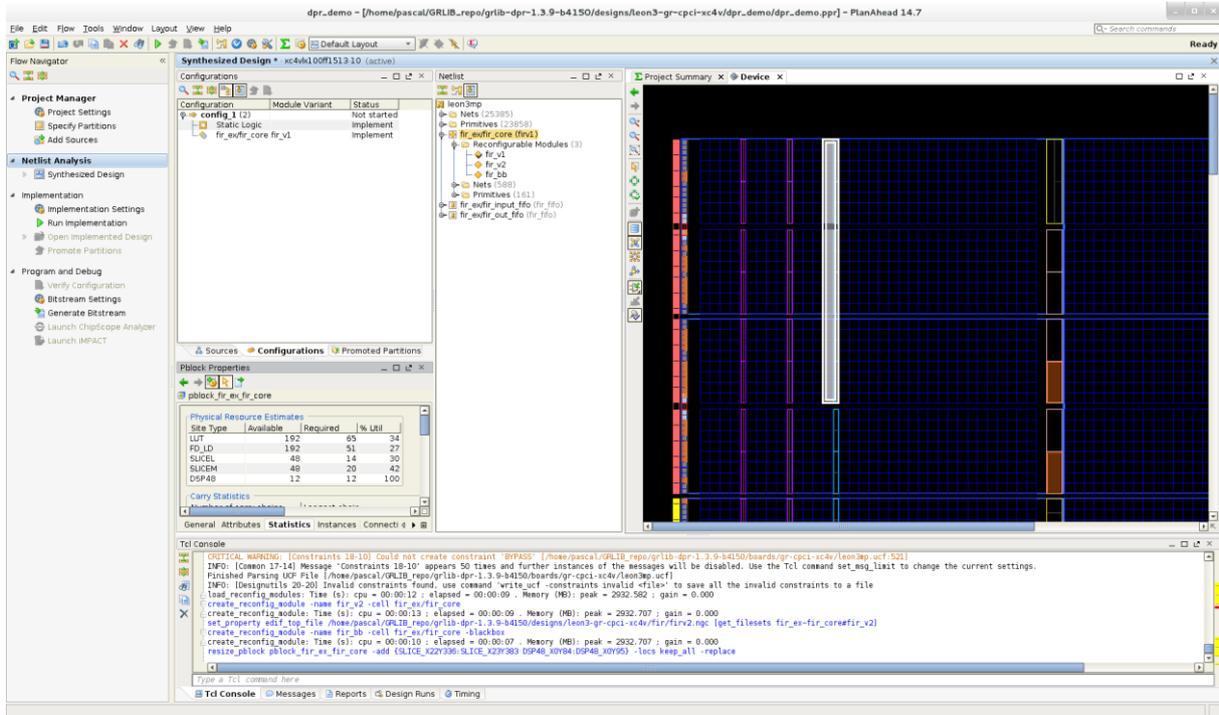


Figure 15. Floorplanning the reconfigurable partition

Set Pblock Size automatically creates one or more AREA_GROUP constraints in the .ucf file:

```
INST "fir_ex/fir_core" AREA_GROUP = "pblock_fir_ex_fir_core";
```

```
AREA_GROUP "pblock_fir_ex_fir_core" RANGE=SLICE_X22Y336:SLICE_X27Y383;
```

```
AREA_GROUP "pblock_fir_ex_fir_core" RANGE=DSP48_X0Y84:DSP48_X0Y95;
```

Consequently, all the resources (including routing) needed by reconfigurable modules will be mapped in the resource range defined by the AREA_GROUP constraint.

Then, the first configuration, including the static design and the first version of the FIR filter (*firv1*), can be implemented. At the end of the implementation process, the tool prompts for **Promoting** implemented partitions. At least the static logic of the design must be promoted, since in the following implementations its place and route information will be imported to ensure compatibility among different configurations.

Looking at *Figure 15*, since the design includes a reconfigurable partition and three reconfigurable modules, two other configurations can be implemented. The first additional configuration includes the static logic and the second version of the FIR filter (*firv2*), while the second one is composed of the static logic and the black-box module (*fir_bb*).

To create new configurations, right-click on the **Design Runs** tab and select **Create New Runs**. In the Create New Runs window (*Figure 16*), select **Partition Action** to define the new configuration. As shown in *Figure 17*, static logic can be imported, while the new selected reconfigurable module must be implemented.

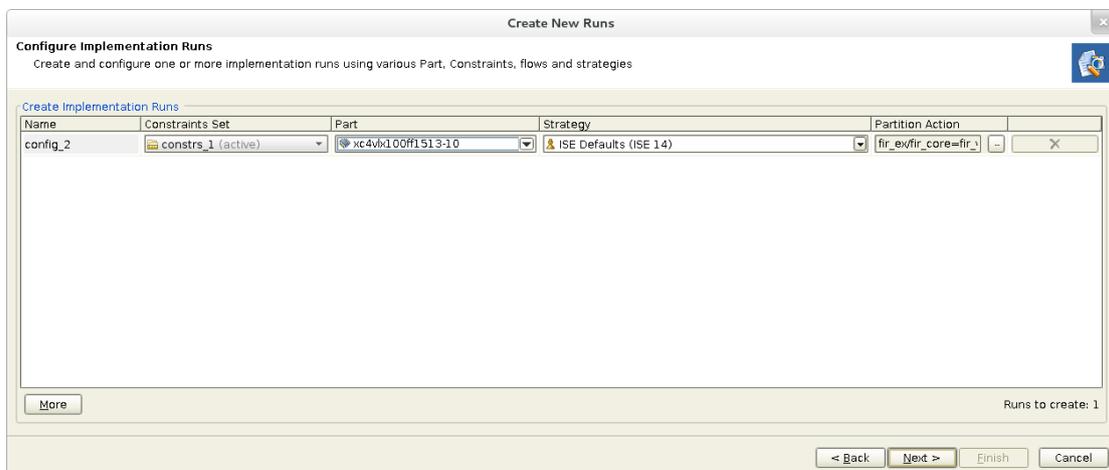


Figure 16. Create New Runs window

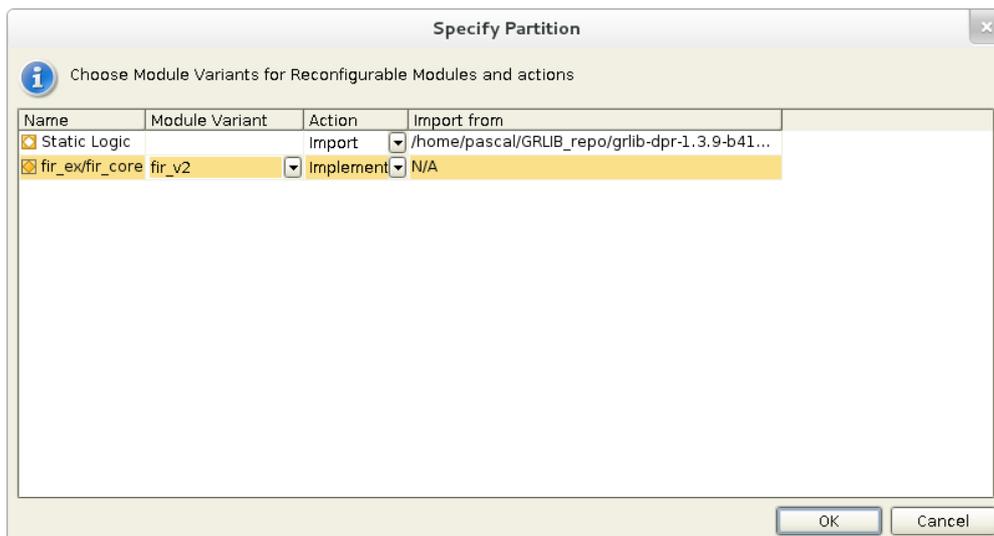


Figure 17. New configuration setup

The two additional configurations can be now implemented. At the end of the implementation process, three configurations are available (Figure 18), each one implementing a different version of the FIR filter instance.

Name	Part	Constraints	Strategy	Host	Status	Progress	Start	Elapsed	Util (%)	FMax (MHz)	Timing Score	Unro
config_1 (active)	xc4vx100ff1513-10	constrs_1	ISE Defaults (IS...	ext...	PAR Complete!	100%	9/19/14 5:02 PM	00:05:22	16.000	61.305	0	0
config_2	xc4vx100ff1513-10	constrs_1	ISE Defaults (ISE 14)	ext2...	PAR Complete!	100%	9/19/14 5:08 PM	00:05:10	16.000	61.305	0	0
config_3	xc4vx100ff1513-10	constrs_1	ISE Defaults (ISE 14)	ext2...	PAR Complete!	100%	9/19/14 5:14 PM	00:04:35	15.000	61.305	0	0

Figure 18. Design Runs window after implementation of three different configurations

Eventually, to ensure static logic compatibility among configurations, the *Verify Configuration* command can be issued. Basically, it verifies that the static logic is implemented in the same way in the three configurations.

2.4 Full and Partial Bitstreams generation

The bitstream generation process produces, for each configuration, two .bit files. The first one represents the full configuration bitstream, while the second one, called *partial bitstream*, includes information needed to configure only the reconfigurable partition.

The FPGA can be initially programmed with one of the three full configuration bitstreams, while, in order to perform dynamic partial reconfiguration, the partial bitstream associated to the actually selected reconfigurable module can be downloaded using DPRC.

To enable partial bitstreams post-processing for DPRC usage, the bitstream generation process must be launched with the *-b* switch set, which produces an ASCII representation of the bitstream data (.rft file).

Although the PlanAhead graphical user interface is fundamental to correctly floorplan the design, its usage is not mandatory. The complete partial reconfiguration design flow can be carried out launching planAhead in batch mode and using a tcl script including all the necessary commands, as in the following example:

```
planahead -mode batch -source pr_project.tcl
```

pr_project.tcl

#create project and add static logic netlists and constraints

```
create_project pr_demo /home/pascal/GRLIB_repo/glib-dpr-1.3.9-b4150/designs/leon3-gr-cpci-xc4v/pr_demo -part xc4vx100ff1513-10
```

```
set_property design_mode GateLvl [current_fileset]
```

```
add_files -norecurse /home/pascal/GRLIB_repo/glib-dpr-1.3.9-b4150/designs/leon3-gr-cpci-xc4v/leon3mp.ngc
```

```
add_files -fileset constrs_1 -norecurse { /home/pascal/GRLIB_repo/glib-dpr-1.3.9-b4150/boards/gr-cpci-xc4v/leon3mp.ucf /home/pascal/GRLIB_repo/glib-dpr-1.3.9-b4150/lib/testgroup/pr/v4v5icap.ucf }
```

```
set_property name config_1 [current_run]
```

```
set_property is_partial_reconfig true [current_project]
```

```
link_design -top leon3mp -name netlist_1
```

#add first reconfigurable module

```
create_reconfig_module -name firv1 -cell fir_ex/fir_core
```

```
set_property edif_top_file /home/pascal/GRLIB_repo/glib-dpr-1.3.9-b4150/designs/leon3-gr-cpci-xc4v/fir/firv1.ngc [get_filesets fir_ex~fir_core#firv1]
```

```
save_constraints
```

```
load_reconfig_modules -reconfig_modules fir_ex/fir_core:fir1
```

#add second reconfigurable module

```
create_reconfig_module -name firv2 -cell fir_ex/fir_core
```

```
set_property edif_top_file /home/pascal/GRLIB_repo/grlib-dpr-1.3.9-b4150/designs/leon3-gr-cpci-xc4v/fir/firv2.ngc [get_filesets fir_ex~fir_core#firv2]
```

#add third reconfigurable module

```
create_reconfig_module -name fir_bb -cell fir_ex/fir_core -blackbox
```

#define Partition constraints

```
resize_pblock pblock_fir_ex_fir_core -add {SLICE_X18Y336:SLICE_X27Y383 DSP48_X0Y84:DSP48_X0Y95} -locs keep_all -replace
```

```
save_constraints
```

#implement first configuration and promote static logic

```
launch_runs config_1 -jobs 8
```

```
wait_on_run config_1
```

```
open_run config_1
```

```
current_design netlist_1
```

```
promote_run -run config_1 -partition_names {leon3mp fir_ex/fir_core}
```

```
create_run config_2 -flow {ISE 14} -strategy {ISE Defaults}
```

#create and implement second configuration

```
config_partition -run config_2 -import -import_dir /home/pascal/GRLIB_repo/grlib-dpr-1.3.9-b4150/designs/leon3-gr-cpci-xc4v/pr_demo/pr_demo.promote/Xconfig_1 -preservation routing
```

```
config_partition -run config_2 -cell fir_ex/fir_core -reconfig_module firv2 -implement
```

```
launch_runs config_2 -jobs 8
```

```
wait_on_run config_2
```

#create and implement third configuration

```
create_run config_3 -flow {ISE 14} -strategy {ISE Defaults}
```

```
config_partition -run config_3 -import -import_dir /home/pascal/GRLIB_repo/grlib-dpr-1.3.9-b4150/designs/leon3-gr-cpci-xc4v/pr_demo/pr_demo.promote/Xconfig_1 -preservation routing
```

```
config_partition -run config_3 -cell fir_ex/fir_core -reconfig_module fir_bb -implement
```

```
launch_runs config_3 -jobs 8
```

```
wait_on_run config_3
```

#verify configurations

```
verify_config -runs {config_1 config_2 config_3} -file pr_verify.log
```

#generate full and partial bitstreams for the three configurations

set_property steps.bitgen.args.d true [get_runs config_1]

set_property steps.bitgen.args.b true [get_runs config_1]

launch_runs config_1 -to_step Bitgen -jobs 8

wait_on_run config_1

set_property steps.bitgen.args.d true [get_runs config_2]

set_property steps.bitgen.args.b true [get_runs config_2]

launch_runs config_2 -to_step Bitgen -jobs 8

wait_on_run config_2

set_property steps.bitgen.args.d true [get_runs config_3]

set_property steps.bitgen.args.b true [get_runs config_3]

launch_runs config_3 -to_step Bitgen -jobs 8

wait_on_run config_3

close_project

exit

3 Partial Reconfiguration Design Flow (Vivado 2015.2)

The steps required by the Partial Reconfiguration design flow shown in the previous section are also valid to implement a partially reconfigurable system on 7-Series FPGAs using Vivado. However, the way commands are implemented differs, since Vivado supports only the non-project mode Partial Reconfiguration design flow, through a tcl console (in this guide it is assumed that all commands are issued using the tcl console included in the Vivado graphical user interface).

Moreover, both netlists and design checkpoints (.dcp) are supported as input files for the PR design flow.

Two classes of design checkpoints must be generated. The first one is associated to the synthesized static logic (not including the reconfigurable module), while the second class includes a design checkpoint for each synthesized reconfigurable module.

To synthesize a reconfigurable module, I/O buffers insertion must be turned off. This can be accomplished by adding the `-mode out_of_context` switch to the `synth_design` command, as example:

```
synth_design -mode out_of_context -flatten_hierarchy rebuilt -top <entity name of the reconfigurable module> -part <device part>
```

A design checkpoint is then created by issuing:

```
write_checkpoint <destination_file_name>.dcp
```

The first design configuration must include the static logic and the first reconfigurable module.

Assuming to have a single design checkpoint for the static logic and an associated constraints file, they can be opened and imported as follows:

```
open_checkpoint <static_logic_checkpoint_filename>.dcp
```

```
read_xdc <static_logic_constraints_filename>.xdc;
```

The first reconfigurable module design checkpoint is then imported and linked to the static logic by issuing:

```
read_checkpoint -cell <reconfigurable_module_instance_name> <reconfigurable_module_checkpoint_filename>.dcp
```

```
set_property HD.RECONFIGURABLE true [get_cells <reconfigurable_module_instance_name>]
```

The last command tells the tool that the chosen instance is a reconfigurable partition.

Physical constraints can now be defined for the reconfigurable partition. The Pblock size can be setup through the Vivado graphical user interface as done in PlanAhead. Right-click on the reconfigurable partition instance, then select **Floorplanning** and click **SetPblock size**.

A limitation, with respect to the Partial Reconfiguration design flow using PlanAhead, is that left and right Pblock edges must be placed between two resource columns and not between interconnect columns. Valid Pblock edges are placed between CLB columns, between a CLB column and a BRAM column, or between a CLB column and a DSP column.

Moreover, partial reconfiguration on 7 Series devices offers the **Reset After Reconfiguration** functionality. This feature leverages the embedded global reset lines of the FPGA to automatically assert a local reset to the reconfigurable module after reconfiguration. To enable the **Reset After Reconfiguration** property, the following command can be issued:

```
set_property RESET_AFTER_RECONFIG true [get_pblocks <pblock_name>]
```

This option can be enabled only if Pblock height spans an entire clock region, as shown in *Figure 19*.

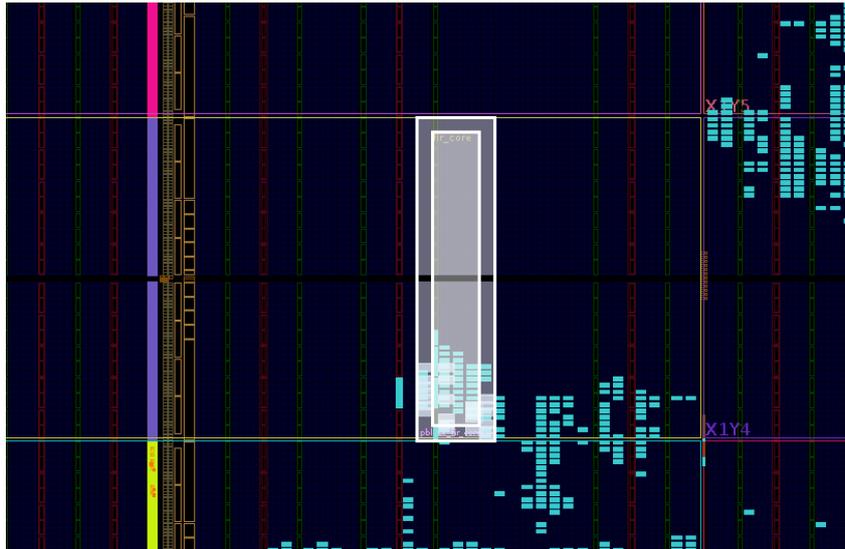


Figure 19. Pblock spanning an entire clock region

To confirm a correct Pblock placement, designer can issue the **Design Rule Check**, which will control for any dynamic partial reconfiguration property violation:

```
report_drc -checks [get_drc_checks HDPR*]
```

To implement the first configuration, and generate the associated full and partial bitstreams, issue the following commands:

```
opt_design
place_design
route_design
write_bitstream -raw_bitfile config1
write_checkpoint config1_routed.dcp
```

Afterwards, in order to implement the second configuration, the static logic routing information must be locked, and the design checkpoint associated to the second reconfigurable module can be imported:

```
update_design -cell <reconfigurable_module_instance_name> -black_box
lock_design -level routing
read_checkpoint -cell <reconfigurable_module_instance_name> <second_reconfigurable_module_checkpoint_filename>.dcp
opt_design
place_design
route_design
write_bitstream -raw_bitfile config2
write_checkpoint config2_routed.dcp
```

These steps can be repeated if design includes more than two reconfigurable modules.

To verify and confirm compatibility of the static logic between different configurations, the **pr_verify** tool can be exploited:

```
pr_verify -full_check config1_routed.dcp config2_routed.dcp -file pr_verify.log
```

4 FPGA run-time reconfiguration using DPRC in a LEON3-based system

To quickly include the partial bitstream files in a C program compiled for the LEON3, a software tool (*dprc_sw*) can be exploited. It is located in `<gplib root>/software/dprc/`. *dprc_sw* takes as inputs one or more raw partial bitstream files (in .rbt format) generated by the Xilinx PlanAhead or Vivado software tools. The raw bitfile can be generated by including the `-b` (PlanAhead) or `-raw_bitfile` (Vivado) option when running the bitstream generation process. The output of *dprc_sw* is a header file. Each partial bitstream is processed and declared in the header as a unidimensional array named `bitstream<i>`, where `<i>` represents the index of the input bitfile.

The software tool can be launched by issuing in a Unix shell:

```
./dprc_sw <path of the input bitstream file 0 (.rbt)> <path of the input bitstream file 1 (.rbt)> ... <path of the input bitstream file N (.rbt)> <Number of 32-bit words per block> <path of the output file (.h)>
```

The software tool can be used also when DPRC is not configured in Dependable CRC mode (see DPRC User Guide).

Either working in *Synchronous*, *Asynchronous* or *D²PRC* mode, DPRC shows five 32-bit registers on the APB address space. To start a partial reconfiguration, assuming that the partial bitstream is already available in memory, the following steps must be executed:

1. Verify if the four least significant bits of the *Status Register* are set to 0xF, meaning that DPRC is available and no errors occurred during the last reconfiguration;
2. Write in the *Address Register* the address of the partial bitstream to be loaded;
3. Set the corresponding bit in the *RM Reset Register* in order to assert a synchronous reset to the reconfigurable module after reconfiguration. Considering the example shown in Section 2.1, since the least significant bit of the *rm_reset* output is used as reconfigurable modules reset, the *RM Reset Register* must be set to 0x00000001;
4. Write the number of 32-bit words composing the partial bitstream to be loaded in the *Control Register*; this starts the reconfiguration process and clears the *Status Register*;
5. Wait until the *Status Register* signals the end of the reconfiguration process or an error (look at the 4 least significant bits).

In addition, the *Timer Register* stores the number of clock cycles spent for the last reconfiguration.

5 Demonstrating Partial Reconfiguration design flow and DPRC usage on GR-CPCI-XC4V reference design (using ISE Design Suite 14.7)

GRLIB leon3-gr-cpci-xc4v template design includes the instantiation of *DPRC* and FIR filter AHB/APB peripheral. A tcl script is also included and can be used as reference for custom user PR designs.

In particular, the template design directory contains the “*dprc_fir_demo*” subdirectory, which includes:

- *fir_ahb_dma_apb.vhd*: implements the top entity of the FIR peripheral, including bus interfaces, DMA and input/output buffers. It does not include the actual filter core, implemented in *fir_v1.vhd* and *fir_v2.vhd*.
- *fir_v1.vhd*, *fir_v2.vhd*: include the description of two FIR filters. Both of them implement the same filtering function. However, *fir_v1.vhd* uses 10 multipliers to produce a valid output value each clock cycle, while *fir_v2.vhd* uses only a multiplier, thus producing 1 valid output data every 10 clock cycles. These two cores represent the two reconfigurable modules.
- *firv1.ngc*, *firv2.ngc*: netlists associated to *fir_v1.vhd* and *fir_v2.vhd*.
- *dprc_demo.tcl*: script implementing the partial reconfiguration design flow in PlanAhead (v14.7), as shown in Section 2.
- *pr_fir_demo.c*: test program source file used to demonstrate *DPRC* usage in a LEON3-based system.

The *leon3mp.vhd* file of the target reference design includes the instantiation of *DPRC* and FIR components as shown in the following statements:

```
-----
--- DYNAMIC PARTIAL RECONFIGURATION -----
-----
prc : if CFG_PRC = 1 generate
p1 : dprc generic map(hindex => CFG_NCPU+CFG_AHB_UART+CFG_GRPIC2_TARGET+CFG_GRPIC2_DMA+log2x(CFG_PCI)+CFG_AHB_JTAG+
CFG_GRETH+CFG_SPW_NUM, pindex => 10+CFG_SPW_NUM, paddr => 10+CFG_SPW_NUM, pirq => 10+CFG_SPW_NUM, technology => CFG_FABTECH,
crc_en => CFG_CRC_EN, edac_en => CFG_EDAC_EN, words_block => CFG_WORDS_BLOCK, fifo_dcm_inst => CFG_DCM_FIFO,
fifo_depth => CFG_DPR_FIFO)
port map(rstn => rstn, clk => clk, clk100 => '0', ahbmi => ahbmi,
ahbmo => ahbmo(CFG_NCPU+CFG_AHB_UART+CFG_GRPIC2_TARGET+CFG_GRPIC2_DMA+log2x(CFG_PCI)+CFG_AHB_JTAG+CFG_GRETH
+CFG_SPW_NUM), apbi => apbi, apbo => apbo(10+CFG_SPW_NUM), rm_reset => rm_reset);
-----
-- FIR component instantiation (for dprc demo) -----
-----
fir_ex : FIR_AHB_DMA_APB
generic map (hindex=>CFG_NCPU+CFG_AHB_UART+CFG_GRPIC2_TARGET+CFG_GRPIC2_DMA+log2x(CFG_PCI)+
CFG_AHB_JTAG+CFG_GRETH+CFG_SPW_NUM+CFG_PRC, pindex=>10+CFG_SPW_NUM+1, paddr=>10+CFG_SPW_NUM+1,
pmask=>16#fff#, technology =>CFG_FABTECH)
port map (rstn=>rstn, clk=>clk, apbi=>apbi, apbo=>apbo(10+CFG_SPW_NUM+1), ahbin=>ahbmi,
ahbout=>ahbmo(CFG_NCPU+CFG_AHB_UART+CFG_GRPIC2_TARGET+CFG_GRPIC2_DMA+log2x(CFG_PCI)+
CFG_AHB_JTAG+CFG_GRETH+CFG_SPW_NUM+CFG_PRC), rm_reset => rm_reset(0));
end generate;
```

fir_ahb_dma_apb component is instantiated only for demonstration purposes, and it should be removed when implementing custom user DPR designs.

After issuing *make xconfig* to setup and enable *DPRC* instantiation (as shown in Section 1), user can issue *make dprc-demo* to synthesize the static logic and execute the entire partial reconfiguration design flow. At the end of the implementation process, *dprc_sw* utility is automatically compiled (using *g++*) and executed to generate the header file storing partial bitstreams for the two filter versions. In addition, the test program is compiled (using *sparc-elf-gcc*) in order to be run on the LEON3 processor.

Issuing *make dprc-demo-prog* will program the FPGA with the first implemented configuration, including the first version of the FIR filter.

Note: make sure that paths to Xilinx tools, *sparc-elf-gcc* and *g++* compilers are properly setup. This demo has been tested using ISE Design Suite 14.7 running on CentOS 7 64-bit.

The test program assumes that *DPRC* is mapped at 0x80000B00 in the APB address space, while FIR peripheral at 0x80000C00.

Connect to the board using *GRMON* and load *dprc_test* program located in <template directory>/dprc_demo/. For each program run, two reconfigurations are triggered. The first reconfiguration swaps the “fast” FIR (*fir_v1.vhd*) with the “slow” FIR (*fir_v2.vhd*), while the second reconfiguration does the opposite. To notice the different behaviour of the two filter versions, look at “Elapsed

Clock Cycles" field, which shows the filtering execution time.

As mentioned in DPRC user guide, it is expected that the first reconfiguration (after power-on) fails with status code 0x8.

The following snippet shows an example of GRMON output.

```
grmon -uart /dev/ttyUSB0 -baud 460800 -u
```

GRMON2 LEON debug monitor

Copyright (C) 2014 Aeroflex Gaisler - All rights reserved.
For latest updates, go to <http://www.gaisler.com/>
Comments or bug-reports to support@gaisler.com

```
Parsing -uart /dev/ttyUSB0  
Parsing -baud 460800  
Parsing -u
```

```
using port /dev/ttyUSB0 @ 460800 baud  
GRLIB build version: 4150  
Detected frequency: 59 MHz
```

Component	Vendor
LEON3 SPARC V8 Processor	Aeroflex Gaisler
AHB Debug UART	Aeroflex Gaisler
Contributed core 1	Various contributions
Contributed core 2	Various contributions
LEON2 Memory Controller	European Space Agency
AHB/APB Bridge	Aeroflex Gaisler
LEON3 Debug Support Unit	Aeroflex Gaisler
Generic UART	Aeroflex Gaisler
Multi-processor Interrupt Ctrl.	Aeroflex Gaisler
Modular Timer Unit	Aeroflex Gaisler
General Purpose I/O port	Aeroflex Gaisler
Generic UART	Aeroflex Gaisler
AHB Status Register	Aeroflex Gaisler

Use command 'info sys' to print a detailed report of attached cores

```
grmon2> info sys  
cpu0 Aeroflex Gaisler LEON3 SPARC V8 Processor  
  AHB Master 0  
ahbuart0 Aeroflex Gaisler AHB Debug UART  
  AHB Master 1  
  APB: 80000700 - 80000800  
  Baudrate 460800, AHB frequency 59.00 MHz  
adev2 Various contributions Contributed core 1 <----- DPRC  
  AHB Master 3  
  APB: 80000B00 - 80000C00  
adev3 Various contributions Contributed core 2 <----- FIR  
  AHB Master 4  
  APB: 80000C00 - 80000D00  
mctrl0 European Space Agency LEON2 Memory Controller  
  AHB: 00000000 - 20000000  
  AHB: 20000000 - 40000000  
  AHB: 40000000 - 80000000  
  APB: 80000000 - 80000100  
  32-bit prom @ 0x00000000  
  64-bit sdram: 2 * 128 Mbyte @ 0x40000000  
  col 9, cas 2, ref 7.8 us  
apbms0 Aeroflex Gaisler AHB/APB Bridge  
  AHB: 80000000 - 80100000  
dsu0 Aeroflex Gaisler LEON3 Debug Support Unit  
  AHB: 90000000 - A0000000  
  AHB trace: 128 lines, 32-bit bus  
  CPU0: win 8, hwbp 2, itrace 128, V8 mul/div, lddel 1  
  stack pointer 0x4ffffff0  
  icache 2 * 16 kB, 32 B/line lrr  
  dcache 2 * 4 kB, 32 B/line lrr  
uart0 Aeroflex Gaisler Generic UART  
  APB: 80000100 - 80000200  
  IRQ: 2  
  Baudrate 38411  
irqmp0 Aeroflex Gaisler Multi-processor Interrupt Ctrl.  
  APB: 80000200 - 80000300  
gptimer0 Aeroflex Gaisler Modular Timer Unit  
  APB: 80000300 - 80000400
```

```

IRQ: 8
8-bit scalar, 3 * 32-bit timers, divisor 59
gpio0 Aeroflex Gaisler General Purpose I/O port
APB: 80000600 - 80000700
uart1 Aeroflex Gaisler Generic UART
APB: 80000900 - 80000A00
IRQ: 3
Baudrate 38411
ahbstat0 Aeroflex Gaisler AHB Status Register
APB: 80000F00 - 80001000
IRQ: 1

```

```

grmon2> load dpr_demo/dpr_test
40000000 .text          132.7kB / 132.7kB [=====>] 100%
400212B0 .data         2.9kB / 2.9kB [=====>] 100%
Total size: 135.54kB (375.11kbit/s)
Entry point 0x40000000
Image /home/pascal/v2GRLIB_repo/grlib-pascal-1.3.9-b4150/designs/leon3-gr-cpci-xc4v/dpr_demo/dpr_test loaded

```

```

grmon2> run
Starting test...
Fast FIR filter results...
825 997 1169 1341 1513 1685 1857 2029 2201 2373 2545 2717 2889 3061 3233 3405 3577 3749 3921 4093 4265 4437 4609 4781 4953 5125 5297 5469 5641
5813 5985 6157 6329 6501 6673 6845 7017 7189 7361 7533 7705 7877 8049 8221 8393 8565 8737 8909 9081 9253 9425 9597 9769 9941 10113 10285 10457
10629 10801 10973 11145 11317 11489 11661 11833 12005 12177 12349 12521 12693 12865 13037 13209 13381 13553 13725 13897 14069 14241 14413
14585 14757 14929 15101 15273 15445 15617 15789 15961 16133 16305
Elapsed clock cycles: 308
Partial Reconfiguration ended...Status:8
Partial Reconfiguration ended...Status:15
Fast FIR filter results...
825 997 1169 1341 1513 1685 1857 2029 2201 2373 2545 2717 2889 3061 3233 3405 3577 3749 3921 4093 4265 4437 4609 4781 4953 5125 5297 5469 5641
5813 5985 6157 6329 6501 6673 6845 7017 7189 7361 7533 7705 7877 8049 8221 8393 8565 8737 8909 9081 9253 9425 9597 9769 9941 10113 10285 10457
10629 10801 10973 11145 11317 11489 11661 11833 12005 12177 12349 12521 12693 12865 13037 13209 13381 13553 13725 13897 14069 14241 14413
14585 14757 14929 15101 15273 15445 15617 15789 15961 16133 16305
Elapsed clock cycles: 321

Program exited normally.

```

```

grmon2> run
Starting test...
Fast FIR filter results...
825 997 1169 1341 1513 1685 1857 2029 2201 2373 2545 2717 2889 3061 3233 3405 3577 3749 3921 4093 4265 4437 4609 4781 4953 5125 5297 5469 5641
5813 5985 6157 6329 6501 6673 6845 7017 7189 7361 7533 7705 7877 8049 8221 8393 8565 8737 8909 9081 9253 9425 9597 9769 9941 10113 10285 10457
10629 10801 10973 11145 11317 11489 11661 11833 12005 12177 12349 12521 12693 12865 13037 13209 13381 13553 13725 13897 14069 14241 14413
14585 14757 14929 15101 15273 15445 15617 15789 15961 16133 16305
Elapsed clock cycles: 308
Partial Reconfiguration ended...Status:15
Slow FIR filter results...
825 997 1169 1341 1513 1685 1857 2029 2201 2373 2545 2717 2889 3061 3233 3405 3577 3749 3921 4093 4265 4437 4609 4781 4953 5125 5297 5469 5641
5813 5985 6157 6329 6501 6673 6845 7017 7189 7361 7533 7705 7877 8049 8221 8393 8565 8737 8909 9081 9253 9425 9597 9769 9941 10113 10285 10457
10629 10801 10973 11145 11317 11489 11661 11833 12005 12177 12349 12521 12693 12865 13037 13209 13381 13553 13725 13897 14069 14241 14413
14585 14757 14929 15101 15273 15445 15617 15789 15961 16133 16305
Elapsed clock cycles: 1230
Partial Reconfiguration ended...Status:15
Fast FIR filter results...
825 997 1169 1341 1513 1685 1857 2029 2201 2373 2545 2717 2889 3061 3233 3405 3577 3749 3921 4093 4265 4437 4609 4781 4953 5125 5297 5469 5641
5813 5985 6157 6329 6501 6673 6845 7017 7189 7361 7533 7705 7877 8049 8221 8393 8565 8737 8909 9081 9253 9425 9597 9769 9941 10113 10285 10457
10629 10801 10973 11145 11317 11489 11661 11833 12005 12177 12349 12521 12693 12865 13037 13209 13381 13553 13725 13897 14069 14241 14413
14585 14757 14929 15101 15273 15445 15617 15789 15961 16133 16305
Elapsed clock cycles: 321

Program exited normally.

```

6 Demonstrating Partial Reconfiguration design flow and DPRC usage on DIGILENT-NEXYS4-DDR reference design (using VIVADO 2015.2)

GRLIB leon3-digilent-nexys4ddr template design includes the instantiation of *DPRC* and FIR filter AHB/APB peripheral. A tcl script is also included and can be used as reference for custom user PR designs using Vivado.

In particular, the template design directory contains the “*dprc_fir_demo*” subdirectory, which includes:

- *fir_ahb_dma_apb.vhd*: implements the top entity of the FIR peripheral, including bus interfaces, DMA and input/output buffers. It does not include the actual filter core, implemented in *fir_v1.vhd* and *fir_v2.vhd*.
- *fir_v1.vhd*, *fir_v2.vhd*: include the description of two FIR filters. Both of them implement the same filtering function. However, *fir_v1.vhd* uses 10 multipliers to produce a valid output value each clock cycle, while *fir_v2.vhd* uses only a multiplier, thus producing 1 valid output data every 10 clock cycles. These two cores represent the two reconfigurable modules.
- *dpr_demo.tcl*: script implementing the partial reconfiguration design flow in Vivado (v2015.2), as described in Section 3.
- *pr_fir_demo.c*: test program source file used to demonstrate *DPRC* usage in a LEON3-based system.

The *leon3mp.vhd* file of the target reference design includes the instantiation of *DPRC* and FIR components as shown in the following statements:

```
-----  
--- DYNAMIC PARTIAL RECONFIGURATION -----  
-----  
prc : if CFG_PRC = 1 generate  
  p1 : dprc generic map(hindex => CFG_NCPU+CFG_AHB_UART+CFG_AHB_JTAG+CFG_GRETH, pindex => 14, paddr => 14, pirq => 14, clk_sel => 1,  
    technology => CFG_FABTECH, crc_en => CFG_CRC_EN, edac_en => CFG_EDAC_EN, words_block => CFG_WORDS_BLOCK,  
    fifo_dcm_inst => CFG_DCM_FIFO, fifo_depth => CFG_DPR_FIFO)  
  port map(rstn => rstn, clk => clk, clk100 => '0', clk100 => sys_clk_i, ahbmi => ahbmi,  
    ahbmo => ahbmo(CFG_NCPU+CFG_AHB_UART+CFG_AHB_JTAG+CFG_GRETH),  
    apbi => apbi, apbo => apbo(14), rm_reset => rm_reset);  
-----  
-- FIR component instantiation (for dprc demo) -----  
-----  
fir_ex : FIR_AHB_DMA_APB  
  generic map (hindex=>CFG_NCPU+CFG_AHB_UART+CFG_AHB_JTAG+CFG_GRETH+CFG_PRC, pindex=>13, paddr=>13,  
    pmask=>16#fff#, technology =>CFG_FABTECH)  
  port map (rstn=>rstn, clk=>clk, apbi=>apbi, apbo=>apbo(13), ahbin=>ahbmi,  
    ahbout=>ahbmo(CFG_NCPU+CFG_AHB_UART+CFG_AHB_JTAG+CFG_GRETH+CFG_PRC), rm_reset => rm_reset(0));  
end generate;
```

fir_ahb_dma_apb component is instantiated only for demonstration purposes, and it should be removed when implementing custom user DPR designs.

After issuing *make xconfig* to setup and enable *DPRC* instantiation (as shown in Section 1), user can issue *make dprc-demo* to synthesize the static logic and execute the entire partial reconfiguration design flow. At the end of the implementation process, *dprc_sw* utility is automatically compiled (using *g++*) and executed to generate the header file storing partial bitstreams for the two filter versions. In addition, the test program is compiled (using *sparc-elf-gcc*) in order to be run on the LEON3 processor.

Issuing *make dprc-demo-prog* will program the FPGA with the first implemented full design configuration, including the first version of the FIR filter.

Note: make sure that paths to Xilinx tools, *sparc-elf-gcc* and *g++* compilers are properly setup. This demo has been tested using Vivado v2015.2 running on CentOS 7 64-bit.

The test program assumes that *DPRC* is mapped at 0x80000E00 in the APB address space, while FIR peripheral at 0x80000D00.

Connect to the board using GRMON and load *dpr_test* program located in <template directory>/dpr_demo/. For each program run, two reconfigurations are triggered. The first reconfiguration swaps the “fast” FIR (*fir_v1.vhd*) with the “slow” FIR (*fir_v2.vhd*), while the second reconfiguration does the opposite. To notice the different behavior of the two filter versions, look at “Elapsed Clock Cycles” field, which shows the filtering execution time.

The following snippet shows an example of GRMON output.

```
> grmon -uart /dev/ttyUSB1 -baud 460800 -u
```

```
Parsing -uart /dev/ttyUSB1
Parsing -baud 460800
Parsing -u
```

Commands missing help:

```
bdump
datacache
```

```
using port /dev/ttyUSB1 @ 460800 baud
GRLIB build version: 4151
Detected frequency: 74 MHz
```

Component	Vendor
LEON3 SPARC V8 Processor	Aeroflex Gaisler
AHB Debug UART	Aeroflex Gaisler
JTAG Debug Link	Aeroflex Gaisler
GR Ethernet MAC	Aeroflex Gaisler
Contributed core 1	Various contributions <----- DPRC
Contributed core 2	Various contributions <----- FIR
AHB/APB Bridge	Aeroflex Gaisler
LEON3 Debug Support Unit	Aeroflex Gaisler
Single-port DDR2 controller	Aeroflex Gaisler
SPI Memory Controller	Aeroflex Gaisler
Generic UART	Aeroflex Gaisler
Multi-processor Interrupt Ctrl.	Aeroflex Gaisler
Modular Timer Unit	Aeroflex Gaisler

Use command 'info sys' to print a detailed report of attached cores

```
grmon2> load ./dpr_demo/dpr_test
40000000 .text          279.0kB / 279.0kB [=====>] 100%
40045BE0 .data         2.9kB / 2.9kB [=====>] 100%
Total size: 281.84kB (372.81kbit/s)
Entry point 0x40000000
Image /home/pascal/150108/grlib-pascal-1.3.9-b4151/designs/leon3-digilent-nexys4ddr/dpr_demo/dpr_test loaded
```

```
grmon2> run
```

Starting test...

Fast FIR filter results...

```
825 997 1169 1341 1513 1685 1857 2029 2201 2373 2545 2717 2889 3061 3233 3405 3577 3749 3921 4093 4265 4437 4609 4781 4953 5125 5297 5469 5641
5813 5985 6157 6329 6501 6673 6845 7017 7189 7361 7533 7705 7877 8049 8221 8393 8565 8737 8909 9081 9253 9425 9597 9769 9941 10113 10285 10457
10629 10801 10973 11145 11317 11489 11661 11833 12005 12177 12349 12521 12693 12865 13037 13209 13381 13553 13725 13897 14069 14241 14413
14585 14757 14929 15101 15273 15445 15617 15789 15961 16133 16305
```

Elapsed clock cycles: 470

Partial Reconfiguration ended...Status:15

Slow FIR filter results...

```
825 997 1169 1341 1513 1685 1857 2029 2201 2373 2545 2717 2889 3061 3233 3405 3577 3749 3921 4093 4265 4437 4609 4781 4953 5125 5297 5469 5641
5813 5985 6157 6329 6501 6673 6845 7017 7189 7361 7533 7705 7877 8049 8221 8393 8565 8737 8909 9081 9253 9425 9597 9769 9941 10113 10285 10457
10629 10801 10973 11145 11317 11489 11661 11833 12005 12177 12349 12521 12693 12865 13037 13209 13381 13553 13725 13897 14069 14241 14413
14585 14757 14929 15101 15273 15445 15617 15789 15961 16133 16305
```

Elapsed clock cycles: 1358

Partial Reconfiguration ended...Status:15

Fast FIR filter results...

```
825 997 1169 1341 1513 1685 1857 2029 2201 2373 2545 2717 2889 3061 3233 3405 3577 3749 3921 4093 4265 4437 4609 4781 4953 5125 5297 5469 5641
5813 5985 6157 6329 6501 6673 6845 7017 7189 7361 7533 7705 7877 8049 8221 8393 8565 8737 8909 9081 9253 9425 9597 9769 9941 10113 10285 10457
10629 10801 10973 11145 11317 11489 11661 11833 12005 12177 12349 12521 12693 12865 13037 13209 13381 13553 13725 13897 14069 14241 14413
14585 14757 14929 15101 15273 15445 15617 15789 15961 16133 16305
```

Elapsed clock cycles: 453

Program exited normally.

7 Additional documents

- [1] “*Partial Reconfiguration User Guide – UG702(v14.5)*”, April 26, 2013 – www.xilinx.com
- [2] “*Vivado Design Suite User Guide – Partial Reconfiguration – UG909(v2015.2)*”, June 24, 2015 – www.xilinx.com