

Isolation Design Flow + Dynamic Function eXchange Example

XAPP1361 (v1.0) June 30, 2021

Summary

This lab application note describes combining Isolation Design Flow (IDF) and Dynamic Function eXchange (DFX) within a single design. With the help of this application note, designers can develop a fail-safe single chip solution, using the Xilinx[®] IDF combined with the Dynamic Function eXchange (DFX), which allows modification of an operating FPGA design, by loading a dynamic configuration file.

This application note follows the rules defined in *Isolation Design Flow for UltraScale+ FPGAs and Zynq UltraScale+ MPSoCs* (XAPP1335) and the DFX rules defined in *Vivado Design Suite User Guide: Dynamic Function eXchange* (UG909). Refer to *Isolation Design Flow for UltraScale+ FPGAs and Zynq UltraScale+ MPSoCs* (XAPP1335) for details on the combined flow of IDF + DFX.

An IDF design example is provided in the *Isolation Design Example for Zynq Ultrascale+ MPSoC Application Note* (XAPP1336). For a DFX example on UltraScale+[™] devices, refer to the *Vivado Design Suite Tutorial*: *Dynamic Function eXchange* (UG947). The document is written with the assumption that the reader is familiar with both IDF and DFX methodologies.

Download the reference design files for this application note from the Xilinx website. For detailed information about the design files, see the Reference Design section.

Introduction

The Xilinx Isolation Design Flow (IDF) is a design methodology that allows for information assurance, functional safety implementations, or any other application module requiring both physical and logical isolation. This methodology is backed by significant schematic analysis and software verification, namely,Vivado[®] Isolation Verifier (VIV), that ensures elimination of single points of failure. Single Chip Crypto (SCC) is one specific application of IDF allowing the implementation of a multichip cryptography system, in a single FPGA or a SoC.

Dynamic Function eXchange (DFX) allows for the reconfiguration of modules within an active design. This flow requires the implementation of multiple configurations, which ultimately results in full bitstreams for the first configuration, and partial bitstreams for each reconfigurable module (RM). The number of configurations required varies by the number of modules that need to be implemented. However, all configurations use the same top-level, or static, placement and routing results. These static results are exported from the initial configuration, and imported by all subsequent configurations, using checkpoints.

Xilinx supports the combined flow of Isolation Design Flow (IDF) and Dynamic Function eXchange (DFX) from Vivado 2020.2 onwards.

Hardware and Software Requirements

The following hardware and software are required for this application note:

- Xilinx ZCU102 Evaluation Board (revision 1.0 or later, with production silicon)
- AC to DC power adapter (12 VDC)
- USB Type-A to Micro-B USB cable for UART communication
- Secure Digital (SD-MMC) card ≤ 32 GB
- Xilinx 2020.2 or later

Note: Future versions of Vivado have not been verified with this application note.

- Integrating LogiCORE SEM IP. Refer to Integrating LogiCORE SEM IP in Zynq UltraScale+ Devices (XAPP1298) and the associated reference design files.
- Xilinx Vitis[™] Unifies Software Development Platform 2020.2 or later

Note: Future versions of Vitis have not been verified with this application note.

- Serial communication terminal application (Tera Term or PuTTY)
- Required associated reference design files that can be downloaded from the Xilinx website.

IDF + DFX

The Isolation Design Flow (IDF) and Dynamic Function eXchange (DFX) are two production solutions from Xilinx. They have been available for Zynq UltraScale+ devices from Vivado 2018.3 onwards. From Vivado 2020.2 onwards, Xilinx supports the combined flow of IDF and DFX. The document is written with the assumption that the reader is familiar with both IDF and DFX methodologies. Refer to *Isolation Design Flow for UltraScale+ FPGAs and Zynq UltraScale+ MPSoCs* (XAPP1335), *Isolation Design Example for Zynq Ultrascale+ MPSoC Application Note* (XAPP1336), *Vivado Isolation Verifier User Guide* (UG1291), *Vivado Design Suite User Guide: Dynamic Function eXchange* (UG909), and *Vivado Design Suite Tutorial: Dynamic Function eXchange* (UG947) for details on these individual methodologies. With this combined support, the user can create nested isolated modules (IM) inside of a reconfigurable partition (RP).

IDF+DFX is enabled by default for all Zynq[®] UltraScale+ MPSoC designs, there is no special param needed to enable combined flow. You need to set a param to enable the appropriate set of design rule checks for IDF+DFX. To enable IDF+DFX DRCs, set the following parameter in Vivado:

set_param hd.enableIDFDRC 1



Implementing the Example Design

This reference design contains two reconfigurable partitions (RP), and under each RP, there is an isolated module (IM). There is one more isolated module in the static region. The design uses a simple bare metal application running in PS to load partial bitstreams. The partial bitstreams are placed into a PS-DDR memory and loaded into the device using an application running on the Cortex[®]-A53. The application uses the **xilfpga** library to load the partial bitstreams through the Processor Configuration Access Port (PCAP). More information on the **xilfpga** library can be found in the *Zynq UltraScale+ MPSoC: Software Developers Guide* (UG1137).

The design static IM contains a MicroBlaze[™] and the ICAP to show isolation capabilities. This lab uses the RTL files from Lab 7 of Vivado Design Suite Tutorial: Dynamic Function eXchange (UG947). This lab operates with the assumption that the user is familiar with the DFX design creation, covered through the labs in Vivado Design Suite Tutorial: Dynamic Function eXchange (UG947).



Figure 1: Design Hierarchy

X25381-060121

You can execute the following steps to implement the example design:

Step 1: Extract the Tutorial Design Files

- 1. Download the reference design files from the Xilinx website.
- 2. Extract the zip file contents to any write-accessible location.

Step 2: Create the Vivado Design and Run Through Synthesis

- 1. Launch Vivado 2020.2 or a later version from the directory where the lab design scripts are located.
- 2. Source the project script from the **Vivado GUI** by running the following command from the **Tcl console**.

```
source ./project_idf_dfx_zcu102.tcl
```

3. Check to see if the IP needs to be updated. Run Reports → Report IP Status and update any out-of-date IP that exists. If updates must be made, use the default setting, which has the core container disabled, but skips the actual synthesis of the IP module. This will be demonstrated during the next step.

	Enable Core Container	8
?	Core Container is a new format for IP that simplifies the directory structure by storing each IP as a single file on disk. It is possible to revert to the old format at any time. For more information on using Core Containers, refer to Vivado Design Suite User Guide: Designing with IP or the Source Management and Revision Control section of the UltraFast Design Methodology Guide for the Vivado Design Suite. Do you wish to convert your existing IP to Core Containers at this time?	n
0) <u>C</u> onvert IP to Core Container and Set as Default in Project	
0	Continue with Core Container Disabled	
	on't show this dialog again	
	ОК	

Figure 2: Update IP Window

4. Click **Generate Block Design** which is under IP INTEGRATOR in the Flow Navigator window. Ensure **Out of context per IP** synthesis option stays selected, and then click **Generate**.

Figure 3: Generate Output Products Window

Generate	Output Products 🛛 🙆 🛞
The following output products	will be generated.
Preview	
Q, ₹ \$	
 Memb_dfx_controller.bd Synthesis Implementation Simulation Hw_Handoff Synthesis Options	(OOC per IP)
) <u>G</u> lobal	
Out of context per IP	
Out of context per <u>B</u> lock	Design
Run Settings	
• On <u>l</u> ocal host:	Number of jobs: 6 🗸 🗸
○ On <u>r</u> emote hosts	Configure <u>H</u> osts
Claunch runs on Cluster	lsf 🗸
Apply	G <u>e</u> nerate Cancel

This step creates all of the IP identified in the block design and launches them through synthesis. This process can take a while to run. For this design, the block design covers the vast majority of the static logic representing the design infrastructure. This step does not launch out-of-context synthesis for the RTL submodules, which includes the shift and count Reconfigurable Modules. The example design instantiates the Zynq UltraScale+ MPSoC devices to support partial bitstream loading, via the PCAP.



Figure 4: Example Design: Block Design

Dynamic Function eXchange Wizard is used to define relationships between the different parts of a DFX design. Within the DFX Wizard, you will define configurations, and configuration runs. A configuration is a full design image, with one RM per RP. A configuration run is a pass through the place and route tools to create a routed checkpoint for that configuration. The DFX Wizard also establishes parent-child relationships between configuration runs, helping automate required parts of the flow, including static design locking and pr_verify , and setting up dependencies between runs, so Vivado knows what steps to rerun when sources are modified. This example project has two configurations and their runs defined. There are two configurations created for the design, **Config1** and **Config2**.

Config1 contains **shift_right** and **count_up** RMs, and Config2 contains **shift_left** and **count_down** RMs. **Synth_1** is the parent run for the design.

There are two runs in the example design. This first run establishes the static design, and the first pair of RMs. The second run uses the locked static design, and the second pair of RMs.

The run **impl_1** is the parent run and implements **Config1**. The child run **child_0_impl_1** implements **Config2**.

Tcl C	Conso	le	Mess	sage	s	Log	Rep	orts	Design Runs	х	
Q,	¥	¢	14	~<	►	»	+	%			
Nam	е						Config	guratior	n C o nstraints		Status
~ Þ	✓ ▷ synth_1 (active)					constrs_1		Not started			
\sim	\triangleright in	pl_1	(active	e)			confi	g_1	constrs_1		Not started
	\triangleright	child	l_0_imp	J_1			config	<u>1</u> 2	constrs_1		Not started
< E	Out-	of-Co	ntext N	1odul	e Ru	ns					

Figure 5: **Design Runs Tab – Configurations and their Runs**



5. Once the mb_dfx_controller block design **Out-of-Context Module Runs** finishes synthesis, click **Run Synthesis** in the Flow Navigator. This launches the synthesis of the remaining modules and the entire example design.

Step 3: Enabling IDF

 After synthesis completion, open the Synthesized Design. As shown in Figure 6, set HD.ISOLATED to true for iso_wrapper, rp_u_count/count, and rp_u_shift/shift. Save the constraints by giving top as a file name.



Figure 6: Enabling IDF

This enables IDF for the $iso_wrapper$ module in the static region and the two modules in the **Config1** RMs.

2. To enable IDF in Config2 RM submodules, the user needs to enable HD.ISOLATED on the synthesis DCPs of the Config2 RMs, and save them. IDF related optimizations takes place in link_design, hence the HD.ISOLATED property must be enabled for RM submodules before link design, for subsequent configurations. To enable HD.ISOLATED on the Config2 RMs, performing the following or this can easily be done, using a post-synth TCL Hook script, as mentioned in the Enable HD.ISOLATED on Config2 RMs using Post-Synth Tcl Hook Script section.



a. From File > Check Point > Open, open shift_left synth dcp by selecting:

```
project_idf_dfx_zcu102/project_idf_dfx_zcu102.runs/shift_left_synth_1/
rp_shift.dcp
```

Choose **No**, when asked to close the project. Set HD.ISOLATED to **true** for **shift**. Save and close the **dcp**.

Figure 7: Enable IDF in RM2 Shift

CHECKPOINT DESIGN * - ×	czu9eg-ffvb1156-2-	e
Netlist x Device Con	straints	? _ 🗆 🖻
품 녀		•
N rp_shift > ⊆ Nets (18) > I shift (rp_shift_shift)		
Cell Properties × Cla	ock Regions	? _ 0 8
I shift		$\leftarrow \rightarrow \phi$
Q ¥ ♦ •€ + FILE_NAME ∨ HD	- 0 24	₽
ISOLATED		
IS_BLACKBOX IS_BOUNDARY_INST		
IS_DEBUGGABLE		× ,
General Properties	Statistics Ne	ts Cell Pin∢ ▶ ≡

b. Similarly, open count down synth dcp from:

project_idf_dfx_zcu102/project_idf_dfx_zcu102.runs/count_down_synth_1/
rp_count.dcp

Set HD.ISOLATED to True for count. Save and close the dcp.



Enable HD.ISOLATED on Config2 RMs using Post-Synth Tcl Hook Script

IMPORTANT! These steps can be executed as an alternative to the previous two steps that enabled IDF, on the shift and count RMs. This step must be performed before launching the Synthesis in the previous section.

1. Create a file with the following contents and name the file: rp1_rm2_post_synth_tcl.tcl.

```
set_property HD.ISOLATED true [get_cells shift] -quiet
write_checkpoint -force -noxdef rp_shift.dcp
```

2. Create one more file with the following contents and name the file: rp2_rm2_post_synth_tcl.tcl.

```
set_property HD.ISOLATED true [get_cells count] -quiet
write_checkpoint -force -noxdef rp_count.dcp
```

3. Add the created files to the project by running the following commands from the Tcl console.

```
add_files -fileset utils_1 -norecurse ./rp1_rm2_post_synth_tcl.tcl
add_files -fileset utils_1 -norecurse ./rp2_rm2_post_synth_tcl.tcl
```

4. Add rp1_rm2_post_synth_tcl.tcl and rp2_rm2_post_synth_tcl.tcl files as **post-synth Tcl Hook** scripts for the corresponding synthesis runs, by running the following commands from the Tcl console.

```
set_property STEPS.SYNTH_DESIGN.TCL.POST [ get_files ./
rp1_rm2_post_synth_tcl.tcl -of
[get_fileset utils_1] ] [get_runs shift_left_synth_1]
set_property STEPS.SYNTH_DESIGN.TCL.POST [ get_files ./
rp2_rm2_post_synth_tcl.tcl -of
[get_fileset utils_1] ] [get_runs count_down_synth_1]
```

5. Launch Synthesis.

Step 4: Enabling IDF+DFX DRCs

By default, IDF+DFX DRCs are disabled. To enable the IDF DRCs, run the following command from the Tcl console:

set_param hd.enableIDFDRC 1

This parameter information is not stored in the database and must be re-enabled for every Vivado session. The parameter is most easily set in Vivado_init.tcl which automatically sets the parameter for each of the Vivado sessions.

Step 5: Floorplanning First Configuration

Open the synth dep for floorplanning the first configuration (Config1). Ensure that the HD.ISOLATED is set to **True** for the following modules, **iso_wrapper**, **rp_u_shift/shift and rp_u_count/count**

EXILINX®

1. Assign package pins and I/O ports for the design. Run the following command from the Tcl console to assign package pins and I/O ports.

```
source ./sources/pins.xdc
```

2. Create Pblocks for Reconfigurable Partitions. Create a Pblock for rp_u_shift and assign site ranges for the Pblock. Run the following command from the Tcl console to create a Pblock for the rp_u_shift.

source ./sources/pb_rp_shift.xdc

3. Similarly, create a Pblock for rp_u_count by running the following command from the Tcl console.

```
source ./sources/pb_rp_count.xdc
```

After running the floorplan, the device looks like Figure 8.



Figure 8: Device View after the RP Floorplan

4. Save the design. Run the DFX DRCs by opening **Reports -> Report DRC...** Select **DFX DRCs**, and click **OK** to run the DRCs.

Figure 9: DFX DRCs Selection Window

	Repor	t DRC	\odot	×						
Check design against selected rule decks and/or individual design rules.										
Results name: Interactive report file: Export to file:	drc_1		()) ;;;							
Apply waivers Display only wai Ignore all waivers Rule Decks	ved violations	Rules (101 of 2416)								
 viadu kule becks default opt_checks placer_check router_check router_check bitstream_ch incr_eco_che eco_checks abs checks 	(S) (S) lecks licks	Q ★ ● > Advwizio (18) ✓ Øpnamic Function eXchange (101) > ✓ Pblock (84) > ✓ Floorplan Analysis (14) > ✓ Abstract shell (3) > SelectIO (1) > MGT (9)								
 ✓ Open in a new tab ? 		ОК	Cancel							

After running the DFX DRCs, you can see there are no warnings or errors. There are two advisory messages. You can ignore those messages as they will resolve after creating the Pblock for the static region which is the *iso_wrapper*.

- 5. Draw Pblocks for the Isolated Modules.
 - a. Draw a Pblock for the static Isolated Module which is the *iso_wrapper*. Run the following command from the Tcl console.

source ./sources/pb_iso_wrapper.xdc

b. Draw a Pblock for the isolated module shift ($shift_right$) inside the Reconfigurable Partition rp_u_shift . Run the following command from the Tcl console:

source ./sources/pb_shift_right.xdc

c. Draw a Pblock for the Isolated Module count (count_up) inside of the Reconfigurable Partitionrp_u_count. Run the following command from the Tcl console:

source ./sources/pb_count_up.xdc

d. Save the constraints by saving the design.

Note: Ignore the constraints order changed warning.

 Run VIV DRCs. These are IDF DRCs which are updated for the IDF+DFX flow. Under Report DRC, select IDF_VIV2-1, IDF_VIV2-2, IDF_VIV2-3 and IDF_VIV2-4. Click OK to run the DRCs.



Figure 10: IDF+DFX DRCs Selection Window

	Report DRC	le 😞					
Check design against selected rule decks and/or individual design rules.							
<u>R</u> esults name:	drc_2	Ø					
Interactive report file:		• • •					
Export to file:		• • •					
	Overwrite ○ Append						
Waivers							
Apply waivers Display only waivers Ignore all waivers	red violations						
Rule Decks	Rules (4 of 2416) 8) a) a) b) cks cks cks cks a) cks ck	(-3)					
✓ Open in a new tab	ОК Са	ncel					

Note: Do not select IDF_VIV2-5 and IDF_VIV2-6 as the design is not implemented yet. After the DRC run is completed you can see many **IDF-4 violations**, as shown in Figure 11.

Figure 11: VIV Errors for PU Adjacency (IDF4_VIV2-4)

Tcl Console Messages	Log Repor	ts Design Runs	DRC	x Package Pins I/O Ports			
Q ≚ ≑ 🕄 🖬	🗹 🌗 297 Erro	irs 🕑 🗿 6 Adviso	ries	Hide All			
Name	Severity ^1	Details					
 All Violations (303) 							
🗸 📬 Isolation (303)							
🗸 📬 Constraints (299)							
IDF_VIV2-4 (297)							
IDF_VIV2 #1	Error	IDF-4: PU adjacency SLICE_X55Y244, SLIC	violatio E_X55Y	n: pblock: pblock_count PU(<u>SLICE_X56Y2+</u> 243, <u>SLICE_X55Y242</u> , <u>SLICE_X55Y241</u> , <u>SLI</u> (
IDF_VIV2 #2	Error	IDF-4: PU adjacency SLICE_X55Y244, SLIC	violatio E_X55Y	n: pblock: pblock_count PU(<u>SLICE_X56Y2+</u> 243, <u>SLICE_X55Y242, SLICE_X55Y241, SLI(</u>			
drc_1 (2 violations) × drc_2 (303 violations) ×							

7. Correct the floorplan violation.

IDF highly recommends taking advantage of the highlighting features of the Vivado tools. The following Tcl script highlights all the Pblocks in the design:

```
Set pblocks [get_pblocks *];set ci 1;foreach pblock $pblocks
{highlight_objects -color_index [expr {1 + ($ci % 19)}] [get_pblocks
$pblock]; incr ci}
```



Shading the Pblocks tells the user what resources are included in it. Although, shading is visible when the Pblock is selected, highlighting it helps for better visibility. Additionally, it helps to differentiate between different Pblocks. In a highlighted Pblock, resources that have color are added to the Pblock, and the regions that are black are not included.

Perform the following to correct the IDF violations.

a. Click the first violation from the DRC report window. This selects the violating Programmable Units (PU) in the device view. Refer to *Isolation Design Flow for UltraScale+ FPGAs and Zynq UltraScale+ MPSoCs* (XAPP1335) for details on PU. You can select multiple violations to see the corresponding violating PUs.



Figure 12: Violating PUs on Device View

b. Zoom to the selected PU area. You can see there is no fence between the nested IM inside of the RP and the static IM in a vertical direction. IPU fence is needed here; the fence can either be inside of the RP or outside of the RP.



Figure 13: Violating PUs in Device View Zoomed

EXILINX®

c. Create a fence inside of the RP. Select the nested IM Pblock, and drag the Pblock edge towards the right side, so that the 1PU fence is created. Or, you can use the following resize_pblock command to change the Pblock boundaries.

resize_pblock pblock_count -add {SLICE_X58Y241:SLICE_X75Y298 BUFCE_LEAF_X312Y16:BUFCE_LEAF_X407Y19 BUFCE_ROW_FSR_X77Y4:BUFCE_ROW_FSR_X102Y4 DSP48E2_X12Y98:DSP48E2_X14Y117 HARD_SYNC_X14Y8:HARD_SYNC_X19Y9 RAMB18_X7Y98:RAMB18_X9Y117 RAMB36_X7Y49:RAMB36_X9Y58} -remove {SLICE_X56Y241:SLICE_X75Y298 BUFCE_LEAF_X304Y16:BUFCE_LEAF_X407Y19 BUFCE_ROW_FSR_X76Y4:BUFCE_ROW_FSR_X102Y4 DSP48E2_X12Y98:DSP48E2_X14Y117 HARD_SYNC_X14Y8:HARD_SYNC_X19Y9 RAMB18_X7Y98:RAMB18_X9Y117 RAMB36_X7Y49:RAMB36_X9Y58} -locs keep_all

After changing the Pblock boundaries, the Pblock loses its coloring. You can rerun the command to color the Pblocks. After creating the fence, the Pblock boundary will be as shown in Figure 14.



Figure 14: Pblock Count Boundary After Creating the Fence

d. Similarly, create an 1PU fence between the static IM Pblock and the nested shift Pblock. Drag the nested Pblock such that the fence is inside of the RP. For the shift block, create a horizontal fence and a vertical fence. You can drag the Pblock boundaries to create the fence or use the following resize_pblock command.

```
resize_pblock pblock_shift -add {SLICE_X58Y300:SLICE_X75Y358
BUFCE_LEAF_X312Y20:BUFCE_LEAF_X407Y23
BUFCE_ROW_FSR_X77Y5:BUFCE_ROW_FSR_X102Y5
DSP48E2_X12Y120:DSP48E2_X14Y141
HARD_SYNC_X14Y10:HARD_SYNC_X19Y11 RAMB18_X7Y120:RAMB18_X9Y141
RAMB36_X7Y60:RAMB36_X9Y70} -remove {SLICE_X56Y300:SLICE_X75Y359
```


BUFCE_LEAF_X304Y20:BUFCE_LEAF_X407Y23 BUFCE_ROW_FSR_X76Y5:BUFCE_ROW_FSR_X102Y5 DSP48E2_X12Y120:DSP48E2_X14Y143 HARD_SYNC_X14Y10:HARD_SYNC_X19Y11 RAMB18_X7Y120:RAMB18_X9Y143 RAMB36_X7Y60:RAMB36_X9Y71} -locs keep_all

The following figure shows the Pblock boundaries after creating the fence.

Figure 15: Pblock Shift Boundary After Creating the Fence



X25384-061421

e. Re-run the VIV DRCs as mentioned in Step 6.

Note: There are no warnings or errors in the DRC report.

8. Save the constraints by saving the design.

Step 6: Set HD.ISOLATED_EXEMPT for Multi-Region/Global Clock Nets

There are a couple of clock nets which are going to multiple isolated modules. Ensure to make them exempt for the IDF rules by applying HD.ISOLATED_EXEMPT on those driver cells. Run the following command from the Tcl console:

source ./sources/iso_exempt.xdc

Save the constraints by saving the design.

Step 7: Implement the First Configuration

1. Launch implementation of the first configuration. Select and right-click **impl_1** from the Design Runs tab, and then click **Launch Step To > route_design.**



Figure 16: Launch route_design for the First Configuration

- 2. Click **Open Implemented Design** after implementation is complete. You can close the **Synthesized design**.
- 3. Run the VIV DRCs for implementation. Open the Report DRC window and select **IDF_VIV2-5** and **IDF_VIV2-6** from the Isolation group. Click **OK** to run these DRCs.





Figure 17: VIV DRCs for Implementation

	Report DRC	S S S
Check design against selec	ted rule decks and/or individual design rules.	4
<u>R</u> esults name:	drc_1	٥
Interactive report file:		
Export to file:		
	● <u>O</u> verwrite ○ <u>A</u> ppend	
Waivers		
Apply waivers		
Display only wai	ed violations	
O Ignore all waivers		
Rule Decks	Rules (2 of 2416)	
Vivado Rule Decks	8) 🔍 🔮 🖨	
🗌 🤗 default	→ → → → → → → → → → → → → → → → → → →	^
Ø opt_checks	> Pblack (11)	
Ø placer_check	Provenance (IDF_VIV2-1)	
☐	s — Constraints (3)	
□ @ incr eco che	cks	2-2)
□ @ eco_checks	Package pin violation (IDF	_VIV2-3)
🗌 🖉 abs_checks	Floorplan violation (IDF_VIV	/2-4)
	✓ ✓ Implementation (2)	m (2) E1
		(-6)
	Physical Configuration (856)	
🖉 Open in a new teb		Ŧ
	r	
	OK	Cancel

You can see there are no warnings or errors in the DRC report.

- Inspect the files in impl_l directory under the Project Runs directory. You will see top_routed_bb.dcp which is a a blackbox dcp and RM dcpsrp_u_shift_shift_right_routed.dcp, rp_u_count_count_up_routed.dcp.
- 5. Close the implemented design. You can now proceed to the second configuration.

Step 8: Implement the Second Configuration

Child_0_impl_1 is the run for the second configuration (Config2). You should use a **pre-opt Tcl Hook script** for floorplanning the second configuration. Run the following commands to floorplan and implement the second configuration. Before proceeding with the implementation of the second configuration, ensure that the RMs of the second configuration has HD.ISOLATED property enabled.

1. Add **child_0_pre_opt_tcl.tcl** script to the project by running the following command from the Tcl console.

```
add_files -fileset utils_1 -norecurse ./sources/child_0_pre_opt_tcl.tcl
```

This file has commands to load XDC constraints which creates RM Pblocks for the second configuration.



2. Add the child_0_pre_opt_tcl.tcl script as pre-opt Tcl Hook script for the run child_0_impl_1 by runnin g the following command from the Tcl console.

```
set_property STEPS.OPT_DESIGN.TCL.PRE [ get_files
child_0_pre_opt_tcl.tcl -of [get_fileset utils_1] ] [get_runs
child_0_impl_1]
```

3. Launch implementation of the second configuration. Select and right-click **child_0_impl_1** from the **Design Runs** tab, and then click **Launch Step To > route_design**.



Broperties		2 0		Display name: Z	ynq UltraScale+	ZCU102	? Evaluati	on Boar	d
Properties		r _ U		Implementation Run Propertie	s Ctrl+E	art0:3.	4		
▷ child_0_impl_1		← →	×	Delete	Delete				
Q 🗎				Make Active		020.2	1118 12	32/ineta	العالد
				Change Run Settings		020.2	1110_12	52/11/510	in oy ii
				Set Incremental Implementation	on	20102	Evaluati	on Boar	d
				Set QoR Suggestions					
				Save As Run Strategy					
				Save As Report Strategy					
				Open Run					
٢				Launch Runs					
General Properties Options	Log Repo	rts Messages	14	Reset Runs		inas			
			»	Launch Next Step: opt_design					
Tcl Console Messages Log	Reports D	esign Runs – × ∣		Launch Step To		6	pt_desig	n	
Q ≚ ≑ I4 ≪ ▶ ≫	+ %		~	Reset to Previous Step		۽ (lace_des	sign	
Name	Configuration	Constraints	1	Generate Bitstream		r	oute_des	sign	TP
v 🚽 synth_1 (active)		constrs_1	1	Display Log		V	vrite_bits	tream	
v v impl_1 (active)	config_1	constrs_1]	Display Reports		0.000	0.010	0.000	0.
child_0_impl_1	config_2	constrs_1		Display Messages					
🗸 🖆 Out-of-Context Module Runs				Shophay Herocageo					
<pre><!-- count_down_synth_1</td--><td></td><td>count_down</td><td>17.</td><td>Copy Run</td><td></td><td></td><td></td><td></td><td></td></pre>		count_down	17.	Copy Run					
<pre> count_up_synth_1</pre>		count_up	1	Create Kuns					
<pre> shift left synth 1</pre>		shift left		Open kun Directory					
			1	Export to Spreadsheet					

4. After the child implementation is complete, click **Open Implemented Design** and select **child_0_impl_1** from the drop-down menu and click **OK** to open the implemented design of Config2.

Figure 19: Open Implemented Design for the Second Configuration

Implementation Completed									
Implementation successfully completed.									
Open Implemented Design child_0_impl_1 Generate Bitstream View Reports	~	,							
Don't show this dialog again									
OK Ca	ncel								

5. Run the VIV DRCs for implementation. Open Report DRC window and select **IDF_VIV2-5** and **IDF_VIV2-6** from the Isolation group. Click **OK** to run these DRCs.

EXILINX®

Note: There are no warnings or errors from the DRC report.

- 6. Close the implemented design.
- 7. Inspect the files in child_0_impl_1 directory under the Project Runs directory. You can see top_routed.dcp which is a full design dcp for the Config2 and RM dcps and is as follows:

```
rp_u_shift_shift_left_routed.dcp, rp_u_count_count_down_routed.dcp
```

Step 9: Running pr_verify

Run the **pr_verify** command from the Tcl console:

```
pr_verify ./project_idf_dfx_zcu102/project_idf_dfx_zcu102.runs/impl_1/
top_routed.dcp
./project_idf_dfx_zcu102/project_idf_dfx_zcu102.runs/child_0_impl_1/
top_routed.dcp
```

You can see there are no errors with the following commands reported in the Tcl console.

```
INFO: [Vivado 12-3253] PR_VERIFY: check points
./project_idf_dfx_zcu102/project_idf_dfx_zcu102.runs/impl_1/top_routed.dcp
and
./project_idf_dfx_zcu102/project_idf_dfx_zcu102.runs/child_0_impl_1/
top_routed.dcp are compatible
```

Step 10: Generating Bitstreams

You will now create the bitstreams, including the full bitstream for Config1 and partial bitstreams for both Config1 and Config2. Run the following command from the Tcl console to create the bitstreams.

```
source ./create_all_bitstreams_zcu102.tcl
```

This creates a new directory called Bitstreams and generates the bitstreams under it.

Step 11: Creating Vitis Application to Load Partial Bitstreams

For this particular example design, partial bitstreams are placed into PS-DDR memory, and loaded into the device using a baremetal application, running on the Cortex-A53. This application takes inputs from the user through a **UART** to select the RM to load. The application uses the **xilfpga** library to load the partial bitstreams through the PCAP. More information on the **xilfpga** library can be found in the*Zynq UltraScale+ MPSoC: Software Developers Guide* (UG1137).

Perform the following steps to create the software application.

- 1. Select File > Export > Export Hardware in Vivado.
- 2. Click Next on the Export hardware platform window.
- 3. Leave the output set to **Pre-synthesis** on the output window, and click **Next**.

EXILINX®

- 4. Leave the XSA file name as the default of **top** and the **Export to** as a location of the project directory. Click **Next** and then **Finish** to build a design image for the Vitis software platform. This creates the **top.xsa** file under the **project_idf_dfx_zcu102** directory.
- 5. Select **Tools** \rightarrow **Launch Vitis IDE**, and the eclipse launcher dialog box appears in Vivado.
- 6. Ensure the workspace maps to the **current project directory** in the eclipse launcher, and then click **Launch** to open the main Vitis GUI.
- 7. Select File → New → Application Project in Vitis.
- 8. Click Next on the welcome screen.
- 9. Select **Create a new platform from hardware (XSA)** tab, and browse to select **top.xsa** to import the file that was exported from Vivado.
- 10. Keep the platform name as **top**, select **Generate boot components**, and select **psu_cortexa53_0**.

-		New Application Project	00
form			-
e: A platfo	orm project will be generated	automatically in workspace for the selected XSA. It can be customized later.	
Select a	a platform from repositor	Create a new platform from hardware (XSA)	
Hardware	Specification		
VSA Eila	wrktysibdoobkup2tuseraver		Browse
ASA File.	/wik/xsjildilobkup2/veelavel		Browse
Boot com	ponents		
🕑 Gene	rate boot components		
Targe	at processor to create FSBL:	• psu_cortexa53_0	
		⊖ psu_cortexr5_0	

Figure 20: **New Application Project > Platform Window**

- 11. Click Next.
- 12. Set dfx_demo as the project name in the Application Project window, and select psu_cortexa53_0 as the target processor. Click Next.



New Application Project									
Application Project Details									
Specify the application project name and its system project properties									
Application project name: dfx_demo									
System Project									
Create a new system pr	piect for the applicat	tion or collect on existing one	rom the workness	•					
create a new system pr	ojection the applica	cion or select an existing one	form the workpsac	-					
Select a system proje	ct	System project details							
Create new									
		System project name:	dfx_demo_syste	m					
		Torget processor							
		larger processor							
		Select target processo	r for the Applicatio	n project.					
		Processor	Associated	applications					
		hier_mb_microblaz	e_0						
		psu_cortexa53_1	dix_demo						
		psu cortexa53 2							
		psu_cortexa53_3							
		psu_cortexr5_0							
		psu_cortexr5_1							
		psu_pmu_0							
		Show all processors in	the hardware energi	ification M	0				
		, Snow all processors in	une naruware spec						
3			< Dock	New S	Cancel	Tinish			
0			< Back	Next >	Cancel	Finish			

13. Keep default values for Domain, and click **Next**.

	New Application	Project		S (S
omain elect a domain for your project or create a new don	nain			•••
elect the domain that the application would link to	or create a new domain			
lote: New domain created by this wizard will have a	ll the requirements of the a	oplication template selected	in the next step	
Select a domain	Domain details			
Create new	Name	etandalona neu cortava52	0	
	Diapley Nome	standalone_psu_cortexa53	_0	
	Display Name:	standalone_psu_cortexa53	_0	
	Operating System:	standalone	•	
	Processor:	psu_cortexa53_0		
	Architecture:	64-bit	•	
	_			

Figure 22: Domain Selection Window

14. Select **Empty Application(C)**, and click **Finish**.



15. Expand **dfx_demo** in the Project Explorer window. Right-click **src** and select **Import Sources**. Browse to the **sources/dfx_demo/src** directory, and click **Open**. Finally, **check all .c and .h** sources in that folder, and click **Finish**.

	Import	Sources	00
File system			P
Import resources	s from the local file system.		
From directory:	/wrk/xsjhdnobkup2/veeraven/idf2/HE	AD/Tests/dsv/HDPR/HIEDE/db	▼ Browse
🖌 🗁 src		🕑 🖻 dfx_demo.c	
		🕑 🖻 dfx_demo.h	
		🗹 🖻 platform_config.h	
		🗹 🖻 platform.c	
		🗹 🖻 platform.h	
Filter Types	Select All Deselect All		
Into folder: dfx	_demo/src		Browse
Options			
Overwrite e>	isting resources without warning		
Create top-l	evel folder		
Advanced >>			
?		Cance	el Finish

Figure 23: Import Sources Window

16. The application uses the xilfpga library to load the partial bitstreams via PCAP. More information on the xilfpga library can be found in the Zynq UltraScale+ MPSoC: Software Developers Guide (UG1137). You must enable xilfpga and dependent libraries in the BSP settings. Select dfx_demo.prj and click Navigate to BSP settings.

Figure 24: Navigate to BSP Settings

File Edit Search Xilinx Project Window Help	_	
🖻 = 🖩 🐚 🛛 🖲 = 🗞 = 🕸 = O = 🔗 = 🖂 S	ø 🖬 (to o + o + 🕴 🚺	
🗣 Explorer 🕱 📄 🖻 📄	👗 dfx_demo_system 🙀 dfx_demo 🕱	
▶ 🖻 dfx_demo.c	🔀 Application Project Settings	
 ▶ m dfx_demo.h ▶ platform config.h 	General	Options
▶ 🖻 platform.c	Project name: <u>dfx_demo</u>	View current BSP settings, or co
 platform.h lscript.ld 	Platform: top	OS/libraries/drivers etc.
README.tx	Runtime: cpp	Navigate to BSP Settings
▶ @≤_ide ☆ dfx_demo.prj	Domain: standalone_psu_cortexa53_0	\wedge
≚ dfx_demo_system.sprj	CPU: psu_cortexa53_0	ប
 Image: point to p	Hardware Specification: View processors, memory r	anges and peripherals.
▶ 👝 hw		
▶ 않 logs ▶ psu_cortexa53_0		
⊱ resources		
> zynqmp_tsbl		
🖌 Assistant 🕱 📄 🕀 🐵 🐔 🔘 🚸 🖇 🚍 🗖		

17. Select Board Support Package under standalone_psu_cortexa53_0 and then click Modify BSP Settings.



Figure 25: Modify BSP Settings Selection

- 18. Select xilfpga, xilsecure, and xilskey libraries from the supported libraries list.
- 19. Select the **xilfpga** from **Overview > standalone** to open configuration for the library. Change secure_mode value to false.



Figure 26: xilfpga Library Settings

Board Support Package	Settings			
Control various settings of	your Board Support Package.			
 ✓ Overview ✓ standalone 	Configuration for library: xi	lfpga		
xilfpga	Name	Value	Default	Туре
xilskey xilsecure ♥ drivers psu_cortexa53_0	base_address data_readback_en debug_mode ocm_address reg_readback_en secure_environment	0x80000 true false 0xfffc0000 true false	0x80000 true false 0xfffc0000 true false	integer boolean boolean integer boolean
	secure_environment secure_mode secure_readback	false false false	true false	boolean boolean boolean

20. Click **OK**.

21. Open dfx_demo_system settings and disable Generate SD card image.

Figure 27: dfx_demo_system Settings

🖳 Explorer 🕱 📄 😫	8 🗖 🗖	≚ dfx_demo_system ☎ 🛠 dfx_demo	
🕶 📰 dfx_demo_system [top]		👗 System Project Settings	
🔻 📋 dfx_demo [standalone_psu_cortexa53_	0]		
🕨 🗊 Includes		General	Options
🔻 🗁 src		Project name: dfx_demo_system	Sysroot:
Idfx_demo.c		Distinguistics and	
Indfx_demo.h		Platform: top	Root FS:
platform_config.h		Runtime: cpp	
platform.c			Kernel Image:
platform.h			Conorate SD cord image
🛐 lscript.ld			Generate SD card image
README.txt		Application Projects	Δ
▶ 💋_ide		Demain Name	Application Desired Name
💘 dfx_demo.prj 🛛 💙		Domain Name	
👗 dfx_demo_system.sprj		standalone_psu_cortexa53_0	dfx_demo
▼ 📑 top (Out-of-date)			
🕨 🗁 export			
🕨 🗁 hw			
🕨 🚌 loas			

22. Build dfx_demo project, and generate dfx_demo.elf. To build the project, select the project from the Explorer drop-down list, and click **Project > Build Project**.



Figure 28: **Build Project**

23. Create a boot image which loads partial Bitstreams into the PS-DDR and Initializes PL with Config1 full Bitstream. Click Xilinx> Create Boot Image. Select Import from existing BIF file option, browse to the directory where you placed the application note files, and then select and open dfx_demo.bif.



Figure 29: Create Boot Image

Create Boot Image					
BIF file already exists at the specified path and will be overwritten with the modified contents. Use 'Preview Bif Changes' button to view the changes in bif contents before overwriting.					
Architecture: Zynq MP	•				
⊖ Create new BIF file ● In	nport from existing BIF file				
Import BIF file path: /wrk/x	sjhdnobkup2/veeraven/idf2/HE	EAD/Tests/ds	/HDPR/HIEDE/dblo_id	f/DIABLO_IDFPRHW/idf_dfx_lat	Browse
Basic					
Output BIF file path: /wrk/	xsjhdnobkup2/veeraven/idf2/H	IEAD/Tests/de	w/HDPR/HIEDE/dblo_i	df/DIABLO_IDFPRHW/idf_dfx_l:	Browse
UDF data:					Browse
Split Output	t format: BIN 🔻			,	
Output path: /wrk/	xsjhdnobkup2/veeraven/idf2/H	IEAD/Tests/d	sv/HDPR/HIEDE/dblo_i	df/DIABLO_IDFPRHW/idf_dfx_l:	Browse
Boot image partitions					
File path	E	ncrypted	Authenticated		
(bootloader) ./project_idf_d	fx_zcu102/top/zynqmp_fsbl n	one	none		Add
./project_idf_dfx_zcu102/to	p/zynqmp_pmufw/pmufw.eli n	one	none		
./Bitstreams/rp_u_count_co	ount_down_partial.bin no	one	none		Delete
./Bitstreams/top_pb_rp_cou	unt_partial.bin no	one	none		Edit
./Bitstreams/rp_u_shift_shif	it_left_partial.bin no	one	none		
./Bitstreams/top_pb_rp_shi	rt_partial.bin no	oné	none		Up
/project_idf_dfx_zcu102/df	x_demo/Debug/dfx_demo.e no	one	none		Down
0		Preview	BIF Changes	Cancel	image

24. Click Create Image and then click OK to overwrite bif file. This creates the BOOT.bin.

Step 12: Running Demo on ZCU102 Board

- 1. Copy the **BOOT.bin** to the ZCU102 SD card, that was created in the previous step.
- 2. **Connect** the UART cable and power cable to the board. Set the **COM port** to an appropriate value for your computer. Set the **Baud Rate** to 115200. **Open a UART terminal** to communicate with the software running on the PS.
- 3. Configure the Board to Boot in **SD-Boot mode** by setting switch SW6 to **1-ON**, **2-OFF**, **3-OFF**, and **4-OFF** as shown in the following figure.



Figure 30: SW6 Switch Settings for SD Boot Mode

4. Load the SD card into the ZCU102 board and power-up the board. You can see the boot image loaded successfully and LEDs are blinking.



Figure 31: Board after BOOT

In the **UART terminal** you can see the software menu to load the partial bitstreams as shown in Figure 32.

Figure 32: Software Menu on UART



5. Enter your **inputs** to load different partial bitstreams. You can see the LEDs pattern changing on the board after each reconfiguration. The following image shows the UART terminal after loading **shift_left** and **count_down** partials.



*** Dynamic Function eXchange SW Trigger ***
Menu 1: Shift Left 2: Shift Right 3: Count Up 4: Count Down 0: Exit > 1
Reconfiguring Shift Left
Partial Reconfiguration Bitstream loaded into the PL successfully
1: Shift Left 2: Shift Right 3: Count Up 4: Count Down 0: Exit > 4
Reconfiguring Count Down
Partial Reconfiguration Bitstream loaded into the PL successfully Menu 1: Shift Left 2: Shift Right 3: Count Up 4: Count Down 0: Exit

Reference Design

This reference design is created using the Vivado Design Suite 2020.2 and the Vitis Unified Software Development Platform 2020.2. Download the reference design files for this application note from the Xilinx website. The reference design files contains RTL source code for the design, and the C files for the Vitis application.

Reference Design Matrix

The following checklist indicates the procedures used for the provided reference design.

Table 1: Reference Design Matrix

Parameter	Description		
General			
Developer name	Satya Pitaka		
Target devices	Zynq UltraScale+ Devices		
Source code provided?	Yes		
Source code format (if provided)	C, Tcl, HDL, Verilog		
Design uses code or IP from existing reference design, application note, 3rd party or Vivado software? If yes, list.	Yes, design reuse (Lab 7) from Vivado Design Suite Tutorial: Dynamic Function eXchange (UG947)		
Hardware Verification			
Hardware verified?	Yes		
Platform used for verification	ZCU102 Evaluation Board		

Conclusion

This application note provides a step-by-step example to combine Isolation Design Flow (IDF) and Dynamic Function eXchange (DFX) on the Zynq UltraScale+ MPSoC devices. This lab highlights the following:

- Enabling IDF on all configurations
- Floorplanning different configurations
- Running VIV DRCs and fixing floorplan violations
- Implementing and generating Bitstreams for different configurations
- Creating a SW application to load partial Bitstreams via PCAP from PS by using Vitis
- Running the demo on ZCU102 and triggering partial reconfiguration using UART console

References

These documents provide supplemental material useful with this guide:

- 1. Isolation Design Flow for UltraScale+ FPGAs and Zynq UltraScale+ MPSoCs (XAPP1335)
- 2. Isolation Design Example for Zynq Ultrascale+ MPSoC Application Note (XAPP1336)

EXILINX®

- 3. Vivado Design Suite Tcl Command Reference Guide (UG835)
- 4. Vivado Design Suite User Guide: Hierarchical Design (UG905)
- 5. Vivado Design Suite User Guide: Dynamic Function eXchange (UG909)
- 6. Vivado Design Suite Tutorial: Dynamic Function eXchange (UG947)
- 7. Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator (UG994)
- 8. Isolation Design Flow available on the Xilinx website.
- 9. Vivado Isolation Verifier User Guide (UG1291)

Revision History

The following table shows the revision history for this document.

Section	Revision Summary	
06/30/2021 Version 1.0		
Initial release. N/A		

Additional Resources and Legal Notices

Xilinx Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see Xilinx Support.

Documentation Navigator and Design Hubs

Xilinx[®] Documentation Navigator (DocNav) provides access to Xilinx documents, videos, and support resources, which you can filter and search to find information. To open DocNav:

- From the Vivado[®] IDE, select Help \rightarrow Documentation and Tutorials.
- On Windows, select Start \rightarrow All Programs \rightarrow Xilinx Design Tools \rightarrow DocNav.
- At the Linux command prompt, enter docnav.

Xilinx Design Hubs provide links to documentation organized by design tasks and other topics, which you can use to learn key concepts and address frequently asked questions. To access the Design Hubs:

- In DocNav, click the Design Hubs View tab.
- On the Xilinx website, see the Design Hubs page.

Note: For more information on DocNav, see the Documentation Navigator page on the Xilinx website.

Please Read: Important Legal Notices

The information disclosed to you hereunder (the "Materials") is provided solely for the selection and use of Xilinx products. To the maximum extent permitted by applicable law: (1) Materials are made available "AS IS" and with all faults, Xilinx hereby DISCLAIMS ALL WARRANTIES AND CONDITIONS, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, OR FITNESS FOR ANY PARTICULAR PURPOSE; and (2) Xilinx shall not be liable (whether in contract or tort, including negligence, or under any other theory of liability) for any loss or damage of any kind or nature related to, arising under, or in connection with, the Materials (including your use of the Materials), including for any direct, indirect, special, incidental, or consequential loss or damage (including loss of data, profits, goodwill, or any type of loss or damage suffered as a result of any action brought by a third party) even if such damage or loss was reasonably foreseeable or Xilinx had been advised of the possibility of the same. Xilinx assumes no obligation to correct any errors contained in the Materials or to notify you of updates to the Materials or to product specifications. You may not reproduce, modify, distribute, or publicly display the Materials without prior written consent. Certain products are subject to the terms and conditions of Xilinx's limited warranty, please refer to Xilinx's Terms of Sale which can be viewed at https:// www.xilinx.com/legal.htm#tos; IP cores may be subject to warranty and support terms contained in a license issued to you by Xilinx. Xilinx products are not designed or intended to be fail-safe or for use in any application requiring fail-safe performance; you assume sole risk and liability for use of Xilinx products in such critical applications, please refer to Xilinx's Terms of Sale which can be viewed at https://www.xilinx.com/legal.htm#tos.

AUTOMOTIVE APPLICATIONS DISCLAIMER

AUTOMOTIVE PRODUCTS (IDENTIFIED AS "XA" IN THE PART NUMBER) ARE NOT WARRANTED FOR USE IN THE DEPLOYMENT OF AIRBAGS OR FOR USE IN APPLICATIONS THAT AFFECT CONTROL OF A VEHICLE ("SAFETY APPLICATION") UNLESS THERE IS A SAFETY CONCEPT OR REDUNDANCY FEATURE CONSISTENT WITH THE ISO 26262 AUTOMOTIVE SAFETY STANDARD ("SAFETY DESIGN"). CUSTOMER SHALL, PRIOR TO USING OR DISTRIBUTING ANY SYSTEMS THAT INCORPORATE PRODUCTS, THOROUGHLY TEST SUCH SYSTEMS FOR SAFETY PURPOSES. USE OF PRODUCTS IN A SAFETY APPLICATION WITHOUT A SAFETY DESIGN IS FULLY AT THE RISK OF CUSTOMER, SUBJECT ONLY TO APPLICABLE LAWS AND REGULATIONS GOVERNING LIMITATIONS ON PRODUCT LIABILITY.

Copyright

© Copyright 2021 Xilinx, Inc. Xilinx, the Xilinx logo, Alveo, Artix, Kintex, Spartan, Versal, Virtex, Vivado, Zynq, and other designated brands included herein are trademarks of Xilinx in the United States and other countries. AMBA, AMBA Designer, Arm, ARM1176JZ-S, CoreSight, Cortex, PrimeCell, Mali, and MPCore are trademarks of Arm Limited in the EU and other countries. All other trademarks are the property of their respective owners.