Tutorial: Reconfigurable DOA Estimation using MUSIC Algorithm via HLS

Introduction

In this lab, you will use Vivado HLS, Vivado IP Integrator and Software Development Kit to create a reconfigurable peripheral using ARM Cortex-A9 processor system on Zynq. You will use Vivado HLS to generate the user-defined IPs, Vivado IPI to create a top-level design which includes the Zynq processor system as a sub-module. During the PR flow, you will define one Reconfigurable Partition which will have two modes. You will create multiple Configurations and run the Partial Reconfiguration implementation flow to generate full and partial bitstreams. You will use Zedboard to verify the design in hardware using a SD card to initially configure the FPGA, and then partially reconfigure the device using the PCAP under user software control.

GitHub Link: https://github.com/Somya-Sharma/SpatialSensing_MUSIC.git

Design Description

The purpose of this lab exercise is to implement a design that can dynamically reconfigure number of active DOA in Spatial sensing using PCAP resource and PS sub-system. This is done for two types of array arrangements, viz. Uniform Linear Array (ULA) and Sparse Array Arrangement (SAA). The proposed reconfigurable architecture for on ZSoC is shown in Fig. ??. The data stored by the ARM Core 0 of the PS in the DDR memory. The AXI Direct Memory Access (DMA) reads the data from DDR memory via memory-mapped AXI Accelerator Coherency Port (ACP), forwards it to spatial sensing block via AXI stream interface and stores the processed data back to the DDR memory. The PS displays the calculated DoA using UART terminal which is used to verify the functionality of the design. The ULA spatial sensing architecture is obtained by removing the SAP block in Fig. ??. The architecture is made reconfigurable in M i.e. number of active transmissions, by DPR based on-the-fly configuration of Extract $\mathbf{V}_n$ and Music Spectra Generation blocks via processor configuration access port (PCAP).

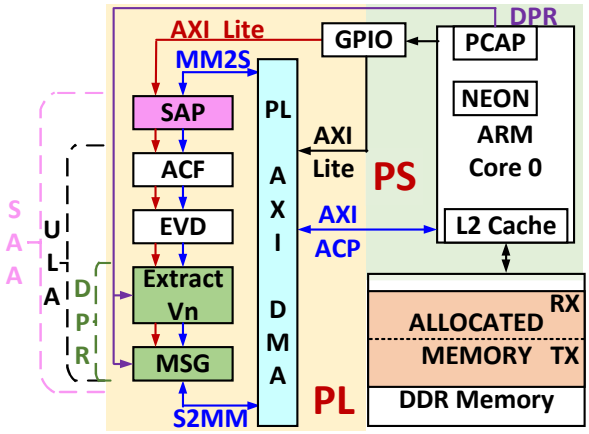
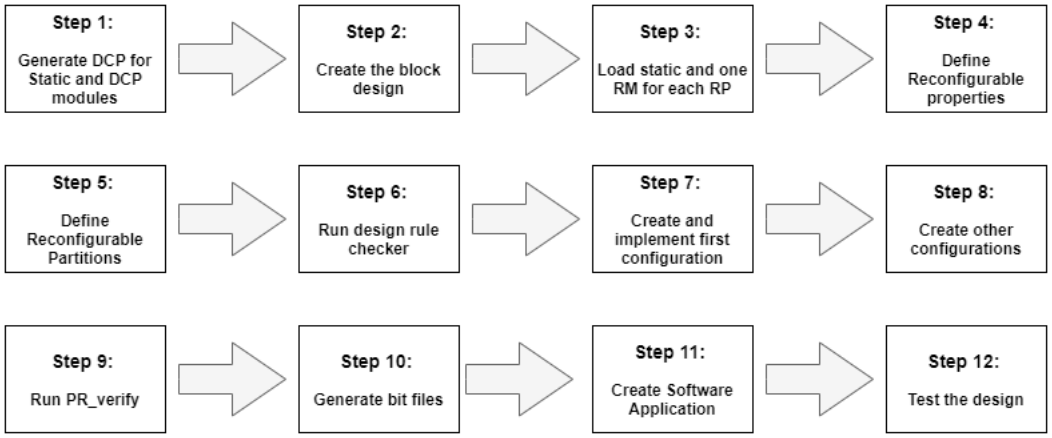


Figure 1: Proposed reconfigurable architecture for spatial sensing on Zynq SoC.

General Flow for this Lab



Procedure

This lab is separated into steps that consist of general overview statements that provide information on the detailed instructions that follow. Follow these detailed instructions to progress through the lab.

Generate DCPs for Static and RM Modules — Step 1

1-1. Start Vivado and execute the provided Tcl script to create the design check point for the static design having one RP.

1-1-1. Open Vivado by selecting **Start** > **All Programs** > **Xilinx Design Tools** > **Vivado 2019.1.**

1-1-2. In the Tcl Shell window enter the following command to change to the lab directory and hit **Enter**.

For SAA, migrate to `C:/DPR_MUSIC/Tutorial/SAA`
For ULA, migrate to `C:/DPR_MUSIC/Tutorial/ULA`

1-1-3. Generate the PS design using given TCl script.

```
source generate_bd.tcl
```

This script will create a new project *DPR_proj* and the block design *bd_dpr*, instantiate ZYNQ PS with SD 0 and UART 1 interfaces enabled. It will also enable the GP0 interface along with FCLK0 and RESET0_N ports. The provided IP for static and reconfigurable region will also be instantiated. It will then create a top-level wrapper file called *bd_dpr_1_wrapper.v* which instantiates the *bd_dpr.bd* (the block design).

1-1-4. Select the **Address Editor** tab. Expand the design heirarchy. Expand **Unmapped Slaves**, if any, and right-click and select **Auto-Assign Address**. Similarly, Expand **Excluded segment**, if any, and right-click and select **Include segment**.

1-1-5. Select **Tools > Validate Design.**

1-1-6. Select **File > Save block design.**

1-2. The current directory contains a folder named Netlists where all the design checkpoints (dcp) will be maintained. It further contains sub folders Static, Config1, Config2, and blank. Synthesize the design to generate the dcp for the static logic of the design.

1-2-1. Click **Run Synthesis** under the Synthesis group in the Flow Navigator to run the synthesis process. Wait for the synthesis to complete. When done click **Cancel**.

1-2-2. Using the windows explorer, copy the *bd_dpr_1_wrapper.dcp* file from *./bd_dpr/bd_dpr.runs/synth_1* into the *./Netlists/Static* directory.

1-2-3. Copy design checkpoints for the *processing_system7_0* instances to the *./Netlists/Static*.

Define the Reconfigurable Partition region — Step 2

2-1. Next you must floorplan the RP region. Depending on the type and amount of resources used by the RMs for the given RP, the RP region must be appropriately defined so it can accommodate any RM variant. For our design, minimum of two clock regions are required for SAA and only one clock region for ULA.

2-1-1. You execute the following TCl script to assign a pblock to the reconfigurable instance and set the properties of hence created pblock.

```
source create_pblock.tcl
```

Load Static and one RM Module — Step 3

Since all required netlist files (dcp) for the design are already given in the Synth folder, you will use Vivado to floorplan the design, define Reconfigurable Partitions, add Reconfigurable Modules, run the implementation tools, and generate the full and partial bitstreams.

3-1. In this step you will load one of the RM designs for the RP.

3-1-1. Since the project is already open, we do not need to load the static design checkpoint. Load the first configuration using the **read_checkpoint** command to the reconfigurable instance.

For SAA, `read_checkpoint -cell [get_cells bd_dpr_i/SAA_Recon_0/inst] ./Netlists/Config1/config1.dcp`

For ULA, `read_checkpoint -cell [get_cells bd_dpr_i/ULA_Recon_0/inst] ./Netlists/Config1/config1.dcp`

3-2. In this design you have only one Reconfigurable Partition. Define the reconfigurable properties to the loaded RM.

3-2-1. Define loaded RM (submodule) as partially reconfigurable by setting the HD.RECONFIGURABLE property using the following command.

For SAA, `set_property HD.RECONFIGURABLE 1 [get_cells bd_dpr_i/SAA_Recon_0/inst]`

For ULA, `set_property HD.RECONFIGURABLE 1 [get_cells bd_dpr_i/ULA_Recon_0/inst]`

This is the point at which the Partial Reconfiguration license is checked.

Create and implement first configuration — Step 4

4-1. Create and implement the first Configuration.

4-1-1. Execute the following command to implement the first configuration. For SAA it is $M = 2$ while for ULA, it is $M = 1$.

```
source config1_route.tcl
```

The script does the following tasks.

- Write the pre-routing design checkpoint for this configuration.

  ```
  write_checkpoint -force ./Netlists/Config1/static1.dcp
  ```

- Optimize, place and route the design by executing the following commands.

  ```
  opt_design

  place_design

  route_design
  ```

- Finally write the post-routing design checkpoint.

  ```
  write_checkpoint -force ./Netlists/Config2/static2_routed.dcp
  ```

4-2. After the first configuration is created, the static logic implementation will be reused for the rest of the configurations. So it should be saved. But before you save it, the loaded RM should be removed.

4-2-1. Execute the following script to update the design with reconfigurable instance as black box.

```
source lock_static.tcl
```

The script will do the following tasks.

- Clear out the existing RMs executing the following commands.

  For SAA, ,`update_design -cell [get_cells bd_dpr_i/SAA_Recon_0/inst] -black_box`

  For ULA, ,`update_design -cell [get_cells bd_dpr_i/ULA_Recon_0/inst] -black_box`

  Issuing this command will result in design changes including, the number of Fully Routed nets (green) decreased, the number of Partially Routed nets (yellow) has increased, and RPs may appear in the Netlist view as empty.
- Lock down all placement and routing by executing the following command.

  ```
  lock_design -level routing
  ```

  Because no cell was identified in the lock_design command, the entire design in memory (currently consisting of the static design with black boxes) is affected.
- Write out the remaining static-only checkpoint by executing the following command.

  ```
  write_checkpoint -force ./Netlists/Static/static_routed_design.dcp
  ```

  This static-only checkpoint would be used for any future configuration, but here, you simply keep this design open in memory.

Create other configurations — Step 5

5-1. Read next RM dcp, create and implement the second configuration.

5-1-1. Execute the following command to create and implement the second configuration. For SAA it is $M = 4$ while for ULA, it is $M = 2$.

For SAA, `read_checkpoint -cell [get_cells bd_dpr_i/SAA_Recon_0/inst] ./Netlists/Config2/config2.dcp`

For ULA, `read_checkpoint -cell [get_cells bd_dpr_i/ULA_Recon_0/inst] ./Netlists/Config2/config2.dcp`

5-1-2. Execute the following TCl script to create the routed design checkpoint for current configuration.

```
source config2_route.tcl
```

The script does following tasks.

- Write the pre-routing design checkpoint for this configuration.

  ```
  write_checkpoint -force ./Netlists/Config2/static2.dcp
  ```

- Optimize, place and route the design by executing the following commands.

  ```
  opt_design

  place_design

  route_design
  ```

- Finally write the post-routing design checkpoint.

  ```
  write_checkpoint -force ./Netlists/Config2/static2_routed.dcp
  ```

Generate bitfiles for created configurations — Step 6

6-1. After all the Configurations have been validated by PR_Verify, full and partial bit files must be generated for the entire project.

6-1-1. The design for the second configuration is already opened. So, there is no need to lead any design checkpoint and you you can directly generate the bitstream for this conifguration.

```
source config2_bitstream.tcl
```

The script does the following tasks.

- Generate the bitstream.

  ```
  write_bitstream -force ./Bitstreams/Config2/config2.bit
  ```

  This command will generate full and partial bitstream for the current configuration.

- Generate the corresponding bin file (of the partial bit file) which will be used to program the FPGA via the SD card.

  ```
  write_cfgmem -force -format BIN -interface SMAPx32 -disablebitswap
  -loadbit "up 0x0 ./Bitstreams/Config2/config2_pblock_inst_partial.bit"
  "./Bin_files/config2.bin"
  ```

6-1-1. Open the design checkpoint for the the first created configuration by executing the below command.

```
open_checkpoint ./Netlists/Config1/static1_routed.dcp
```

6-1-2. Execute the following TCL script in the new window that opens.

```
source config1_bitstream.tcl
```

The script does the following tasks.

- Generate the bitstream.

  ```
  write_bitstream -force ./Bitstreams/Config1/config1.bit
  ```

- Generate the corresponding bin file which will be used to program the FPGA via the SD card.

  ```
  write_cfgmem -force -format BIN -interface SMAPx32 -disablebitswap
  -loadbit "up 0x0 ./Bitstreams/Config1/config1_pblock_inst_partial.bit"
  "./Bin_files/config1.bin"
  ```

- Close the project.

  ```
  close_project
  ```

Create blank configuration — Step 7

7-1. Create the blanking configuration.

7-1-1. Open the design checkpoint for the the blank configuration (which is routed static region) executing the below command.

```
open_checkpoint ./Netlists/Static/static_routed.dcp
```

7-1-2. Execute the following Tcl script to create the blank configuration i.e. no module is loaded to the reconfigurable partition.

```
source blank_bitstream.tcl.
```

The script does the following tasks.

- For creating the blanking configuration, use the update_design -buffer_ports command to insert LUTs tied to constants to ensure the outputs of the reconfigurable partition are not left floating.

  For SAA, `update_design -buffer_ports -cell bd_dpr_i/SAA_Recon_0/inst`
  For ULA, `update_design -buffer_ports -cell bd_dpr_i/ULA_Recon_0/inst`

- Now place and route the design. There is no need to optimize the design.

  ```
  place_design
  ```

  ```
  route_design
  ```

  The base (or blanking) configuration bitstream, when we generate in the next section, will have no logic for either reconfigurable partition, simply outputs driven by ground. Outputs can be tied to VCC if desired, using the HD.PARTPIN_TIEOFF property.

- Save the checkpoint in the BLANK directory.

  ```
  write_checkpoint -force ./Netlists/Blank/blank_routed.dcp
  ```

Export the hardware and open SDK — Step 8

8-1. Export the architecture as a hardware definition file.

8-1-1. Execute the following TCl script for this step.

```
source exportHW.tcl
```

This script does the following tasks

- Create the .sdk folder to where the user application will be created.

  For SAA, `file mkdir C:/DPR_MUSIC/Tutorial/SAA/DPR_proj/DPR_proj.sdk`
  For ULA, `file mkdir C:/DPR_MUSIC/Tutorial/ULA/DPR_proj/DPR_proj.sdk`

- Write the hardware definition file.

  For SAA, `write_hwdef -force -file C:/DPR_MUSIC/Tutorial/SAA/DPR_proj/DPR_proj.sdk/bd_dpr_wrapper.hdf`
  For ULA, `write_hwdef -force -file C:/DPR_MUSIC/Tutorial/ULA/DPR_proj/DPR_proj.sdk/bd_dpr_wrapper.hdf`

You can also go to **File** > **Export Hardware**, leave the **Include Bitstream** option **unchecked**, click **OK** for this step.

8-1-2. Select **File** > **Launch SDK**

8-1-3. Click OK to launch SDK. The SDK program will open. Close the Welcome tab if it opens

Create the software application — Step 9

9-1. Create a Board Support Package enabling generic FAT file system library.

9-1-1. In SDK, select **File** > **New** > **Board Support Package**.

9-1-2. Click **Finish** with the default settings (with standalone operating system). This will open the Software Platform Settings form showing the OS and libraries selections.

9-1-3. Select **xilffs** as the FAT file support is necessary to read the partial bit files from the SD card.

9-1-4. Click OK to accept the settings and create the BSP.

9-2. Create an application.

9-2-1. Select **File** > **New** > **Application project**.

9-2-2. Enter TestApp as the Project Name, and for Board Support Package, choose Use Existing (standalone_bsp_0 should be the only option). If this option does not appear, make sure that the hardware specification is set to the hdf created above in step 8-1-1.

9-2-3. Click **Next**, and select *Empty Application* and click **Finish**.

9-2-4. Expand the **TestApp** entry in the project view, right-click the *src* folder, and select **Import**.

9-2-5. Expand **General category** and double-click on **File System.**.

9-2-6. Browse to C:/DPR_MUSIC/Tutorial/Sources/<SAAorULA> or and click **OK**.

9-2-7. Select *TestApp.c, lib_music.h, lib_music.c, pcap.h, pcap.c, sdCard.h, sdCard.c, platform.h, platform.c, platform_config.h* and **click Finish** to add the file to the project.

9-2-8. Right-click on **TestApp** and select **C/C++ Building Settings**.

9-3. Create a zynq_fsbl application.

9-3-1. Select **File** > **New** > **Application project**.

9-3-2. Enter **zynq_fsbl** as the Project Name, and for Board Support Package, choose **Create New**.

9-3-3. Click **Next**, select **Zynq FSBL**, and click **Finish**.
This will create the first stage bootloader application called **zynq_fsbl.elf**

9-4. Create a Zynq boot image.

9-4-1. Select **Xilinx Tools** > **Create Boot Image**.

9-4-2. Click the Browse button of the Output BIF file path field, browse to *C:/DPR_MUSIC/Tutorial/<SAAorULA>* and then click **Save** with the output as the default filename.

9-4-3. Click on the **Add** button of the Boot image partitions, click the Browse button in the Add Partition form, browse to *C:/DPR_MUSIC/Tutorial/<SAAorULA>/DPR_proj/DPR_proj.sdk/zynq_fsbl/Debug* select **zynq_fsbl.elf** and click **Open**.

9-4-4. Click **OK**. Click again on the **Add** button of the Boot Image partitions, click the Browse button in the Add Partition form, browse to **C:/DPR_MUSIC/<SAAorULA>/Bitstreams** directory, select BLANK.bit and click **Open**.

9-4-5. Click **OK**.

9-4-6. Click again on the **Add** button of the Boot Image partitions, click the Browse button in the Add Partition form, browse to **C:/DPR_MUSIC/Tutorial/<SAAorULA>/DPR_proj/DPR_proj.sdk/TestApp/Debug**, select **TestApp.elf** and click **Open**.

9-4-7. Click **OK**.

9-4-8. Make sure that the output path is **C:/DPR_MUSIC/Tutorial/<SAAorULA>/DPR_proj/** and the filename is **BOOT.bin**. Click **Create Image**.

9-4-9. Close the SDK program by selecting **File** > **Exit**.

Test the design — Step 10

10-1. Connect the board with micro-USB cable connected to the UART. Place the board in the SD boot mode. Copy the generated BOOT.bin and the partial bit files on the SD card, and place the SD card in the board. Power On the board.

10-1-1. Make sure that a micro-usb cable is connected to the UART port.

10-1-2. Make sure that the board is set to boot in SD card boot mode. For this, check that the jumper connections are set to program from the SD card.

10-1-3. Using the Windows Explorer, copy the **BOOT.bin** and other partial binaries on to a SD Card.

10-1-4. Place the SD Card in the board and power ON the board.

10-2. Start a terminal emulator program such as TeraTerm or HyperTerminal. Select an appropriate COM port (you can find the correct COM number using the Control Panel). Set the COM port for 115200 baud rate communication.

10-2-1. Start a terminal emulator program such as **TeraTerm** or **Terminal**.

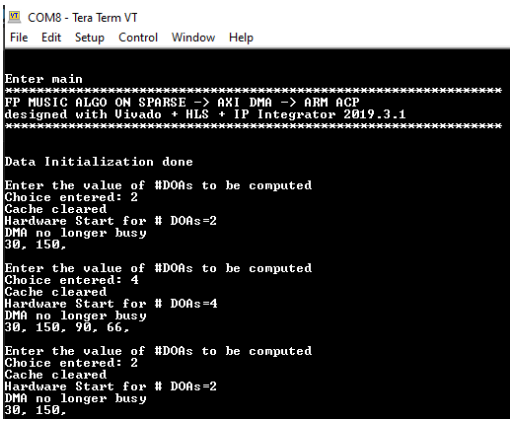
10-2-2. Select the appropriate COM port (you can find the correct COM number using the Control Panel).

10-2-3. Set the COM port for **115200** baud rate communication.

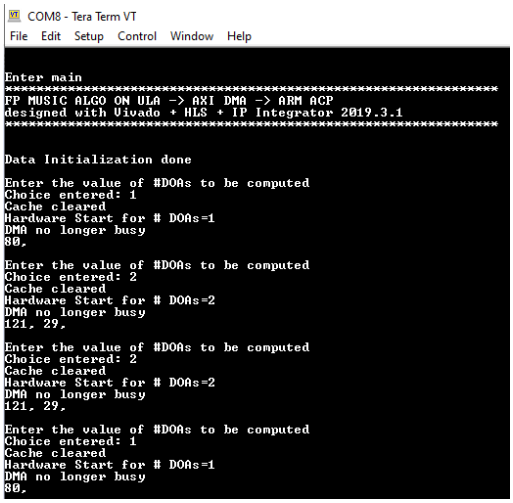
10-2-4. Press BTN7 to display a menu.

10-2-5. Choose either 2 or 4 for SAA and 1 or 2 for ULA.

10-2-6. Below in Fig. 2 is an example user test to show how the terminal window will appear after various reconfigurations.



(a) SAA



(b) ULA

Figure 2: DPR Result for ULA and SAA