Tutorial: Intelligent & Reconfigurable Wireless Physical Layer (PHY)

Introduction

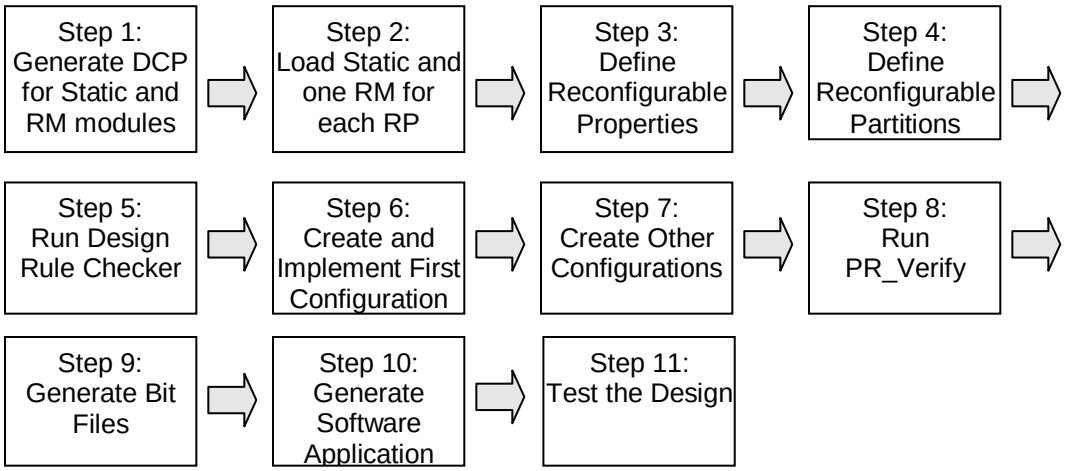
In this lab, you will use Vivado IPI and Software Development Kit to create a reconfigurable peripheral using ARM Cortex-A9 processor system on Zynq. You will use Vivado IPI to create a top-level design, which includes the Zynq processor system as a sub-module. During the PR flow, you will define four Reconfigurable Partitions (MAB, PHY) having Reconfigurable Modules (UCB, UCB_V, UCB_T), (16-QAM, QPSK) respectively. You will create multiple Configurations and run the Partial Reconfiguration implementation flow to generate full and partial bitstreams. You will use ZC706 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/Sai-Santosh-99/ReconfigPHY

Design Description

The purpose of this lab exercise is to implement a design that can be intelligent & dynamically reconfigurable Wireless Physical Layer using PCAP resource and PS sub-system. The system consists of four Online Machine Learning peripherals (for 4 channels) having three unique function calculation capabilities (UCB_T, UCB_V, UCB), one Wireless Transmitter PHY having two modulation capabilities (16-QAM, QPSK), one Wireless Receiver PHY having two demodulation capabilities (for 16-QAM, for QPSK) and a reconfigurable Comparator design. The architecture is designed such that the No. of channels as well as the OML algorithm can be reconfigured in the MAB block on-the-fly. Also, the modulation & demodulation blocks inside the Transmitter PHY & Receiver PHY respectively can also be configured on-the-fly based on the channel characteristics learnt by the MAB block. The Transmitter PHY selects the channel for transmission based on the input it receives from the MAB block. The Transmitter PHY then transmits the bits to the receiver where the pilot power ration is calculated. This ratio is then passed to the MAB block using the AXI-Lite interface. The MAB block then uses this ratio as the reward for the particular channel used to transmit the data. It then calculates the Q-Factor for all channels and then selects the channel with the maximum value to be used for transmission in the next window. The user verifies the functionality using a user application. The dynamic modules are reconfigured using the PCAP resource available through Device Configuration block. The design is shown in Figure 1.

General Flow for this Lab



Design

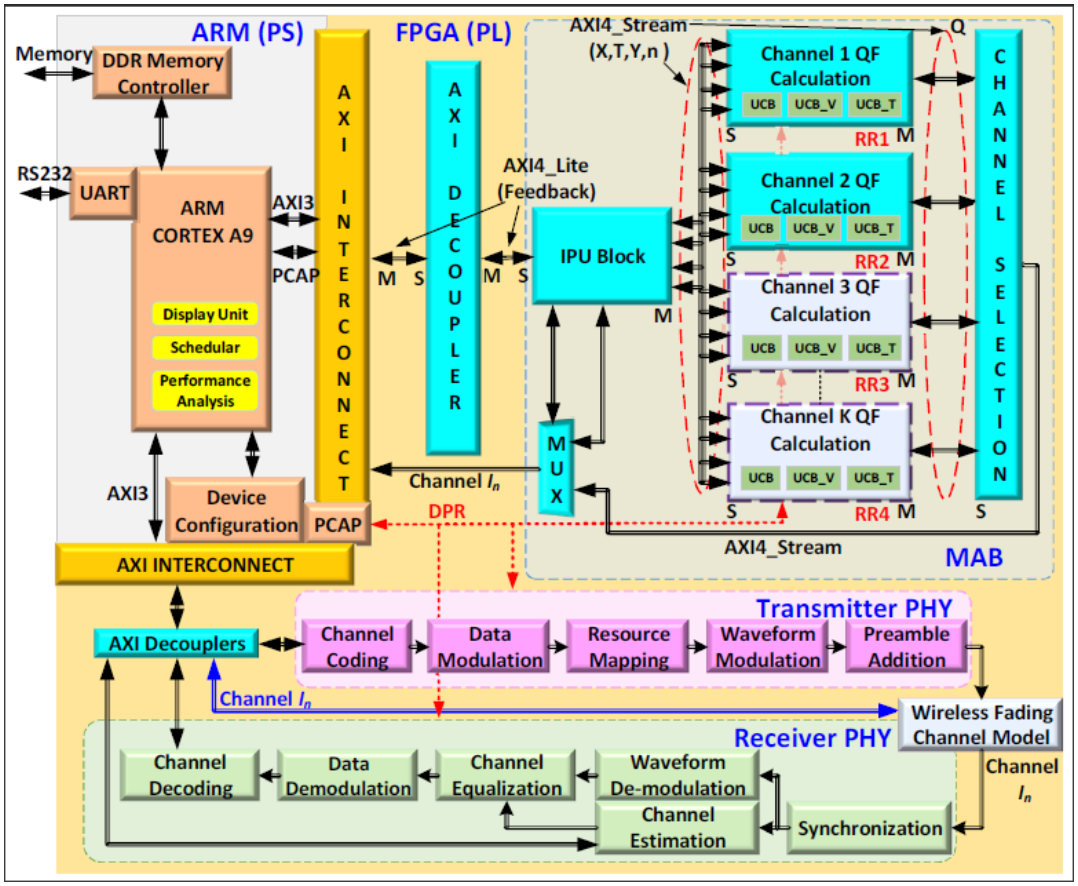


Figure 1. The design

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 the Static Design 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 2018.2 > Vivado 2018.2**

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

```
cd c:/Summer/Tutorial
```

1-1-3. Generate the PS design executing the provided Tcl script.

```
source blockDesign.tcl
```

This script will create the block design called system, 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 OFDM IP, MAB IPs, TRANSFER IPs, & COMPARE IPs will also be

instantiated. It will then create a top-level wrapper file called design_1_wrapper.v which instantiates the design_1.bd (the block design).

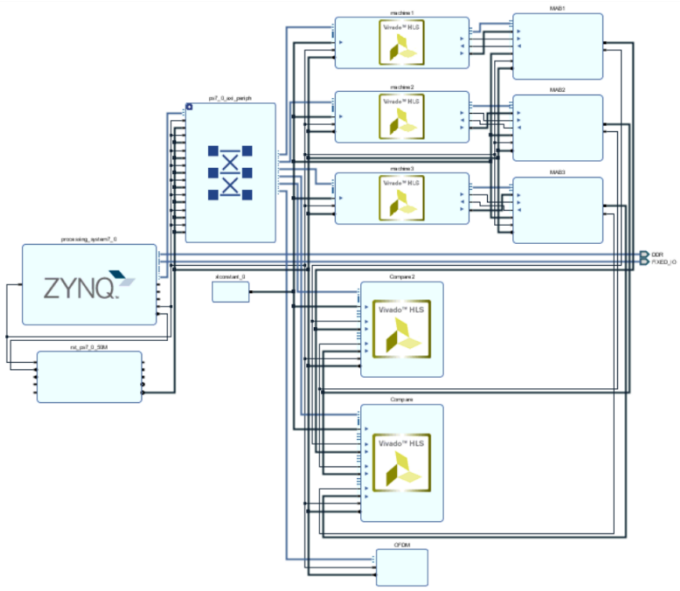


Figure 2. The system block design

1-1-4. Select the **Address Editor** tab. Expand **Data**. Expand **Unmapped Slaves**, if any, and right-click and select **Assign Address.**

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

1-1-6. Select **File** > **Save Block Design.**

1-2. 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 **design_1_wrapper.dcp** file from *tutorial\tutorial.runs\synth*_1 into the *Synth\Static* directory under the current lab directory.

1-2-3. Copy design checkpoints for the auto_pc, informTransfer_0, informTransfer_1, informTransfer_2, streamBlank_0, streamBlank_1, streamBlank_2, DPR_QAM_QPSK_0, compare_0, comparePR_0, xbar_0, rst_ps7_0_50M, and processing_system7_0 instances to *Synth\Static* to sit alongside system_wrapper.dcp

1-2-4. Close the project by typing the `close_project` command in the Tcl console or selecting **File > Close Project**.

Load Static and one RM for the RP in Vivado — Step 2

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.

2-1. In this step you will load the static and one RM designs for the RP.

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

```
cd c:/Summer/Tutorial
```

2-1-2. Execute the following Tcl script to load the static design checkpoint.

```
source load_design_checkpoints.tcl
```

The script will do the following:

- Load the static design using the **open_checkpoint** command.

  ```
  open_checkpoint Synth/Static/design_1_wrapper.dcp
  ```

- Load the IP checkpoint for the Processing System by using the **read_checkpoint** command.

  ```
  read_checkpoint -cell design_1_i/processing_system7_0
  Synth/Static/system_processing_system7_0_0.dcp
  ```

- Load the IP checkpoint for the Processing Reset by using the **read_checkpoint** command.

  ```
  read_checkpoint -cell system_i/rst_ps7_0_50M
  Synth/Static/system_rst_ps7_0_50M_0.dcp
  ```

- Load the IP checkpoint for the auto pc by using the **read_checkpoint** command.

  ```
  read_checkpoint -cell
  system_i/ps7_0_axi_periph/s00_couplers/auto_pc
  Synth/Static/system_auto_pc_0.dcp
  ```

- Load the IP checkpoint for the system bus xbar by using the **read_checkpoint** command.

  ```
  read_checkpoint -cell system_i/ps7_0_axi_periph/xbar
  Synth/Static/system_xbar_0.dcp
  ```

- Load the IP checkpoint for the MAB by using the **read_checkpoint** command.

  ```
  read_checkpoint -cell design_1_i/MAB1
  Synth/Static/design_1_streamBlank_1_0.dcp

  read_checkpoint -cell design_1_i/MAB2
  Synth/Static/design_1_streamBlank_2_0.dcp

  read_checkpoint -cell design_1_i/MAB3
  Synth/Static/design_1_streamBlank_0_0.dcp
  ```

- Load the IP checkpoint for the Transfer IP by using the **read_checkpoint** command.

  ```
  read_checkpoint -cell design_1_i/machine1
  Synth/Static/design_1_informTransfer_0_0.dcp

  read_checkpoint -cell design_1_i/machine2
  Synth/Static/design_1_informTransfer_1_0.dcp

  read_checkpoint -cell design_1_i/machine3
  Synth/Static/design_1_informTransfer_2_0.dcp
  ```

- Load the IP checkpoint for the Compare by using the **read_checkpoint** command.

  ```
  read_checkpoint -cell design_1_i/Compare
  Synth/Static/design_1_compare_0_0.dcp

  read_checkpoint -cell design_1_i/Compare2
  Synth/Static/design_1_comparePR_0_0.dcp
  ```

- Load the IP checkpoint for the OFDM Transceiver by using the **read_checkpoint** command.

```
read_checkpoint -cell design_1_i/OFDM
Synth/Static/design_1_DPR_QAM_QPSK_0_0.dcp
```

2-1-3. Load one RM (UCB) for the MAB RP & two RMs (QAM modulation & QAM demodulation) by using the **read_checkpoint** command.

```
read_checkpoint -cell design_1_i/MAB1/inst/u1 Synth/UCB/ucb_synth.dcp

read_checkpoint -cell design_1_i/MAB2/inst/u1 Synth/UCB/ucb_synth.dcp

read_checkpoint -cell design_1_i/MAB3/inst/u1 Synth/UCB/ucb_synth.dcp

read_checkpoint -cell
design_1_i/OFDM/inst/DPR_QAM_QPSK_v1_0_S00_AXI_inst/T1/UUT/DAT_Mod_Ins
Synth/QAM/mod_synth.dcp

read_checkpoint -cell
design_1_i/OFDM/inst/DPR_QAM_QPSK_v1_0_S00_AXI_inst/T1/UUT1/DataSymDem_
ins Synth/QAM/demod_synth.dcp
```

Define Reconfigurable Properties on each RM Step 3

3-1. In this design you have five Reconfigurable Partitions. Define the reconfigurable properties to the loaded RMs.

3-1-1. Define each of the loaded RMs (submodules) as partially reconfigurable by setting the **HD.RECONFIGURABLE** property using the following commands.

```
set_property HD.RECONFIGURABLE 1 [get_cells
design_1_i/OFDM/inst/DPR_QAM_QPSK_v1_0_S00_AXI_inst/T1/UUT/DAT_Mod_Ins]

set_property HD.RECONFIGURABLE 1 [get_cells
design_1_i/OFDM/inst/DPR_QAM_QPSK_v1_0_S00_AXI_inst/T1/UUT1/DataSymDem_
ins]

set_property HD.RECONFIGURABLE 1 [get_cells design_1_i/MAB1/inst/u1]

set_property HD.RECONFIGURABLE 1 [get_cells design_1_i/MAB2/inst/u1]

set_property HD.RECONFIGURABLE 1 [get_cells design_1_i/MAB3/inst/u1]
```

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

Define the Reconfigurable Partition Region

3-2. Next you must floorplan the RP regions. Depending on the type and amount of resources used by all the RMs for the given RP, the RP region must be appropriately defined so it can accommodate any RM variant.

3-2-1. You execute the following command to define the region for each RP, perform the DRC.

```
read_xdc fplan.xdc
```

Create and Implement First Configuration

4-1. Create and implement the first Configuration.

4-1-1. Execute the following command to implement the first configuration, the UCB algorithm inside the MAB block with QAM modulation scheme for the transceiver.

```
source create_first_configuration.tcl
```

The script will do the following tasks:

- The script will optimize, place and route the design by executing the following commands.

  ```
  opt_design

  place_design

  route_design
  ```

- Save the full design checkpoint.

  ```
  write_checkpoint -force Implement/UCB_QAM/top_ucb_qam_synth.dcp
  ```

  At this point, a fully implemented partial reconfiguration design from which full and partial bitstreams can be generated is ready. The static portion of this configuration **must** be used for all subsequent configurations, and to isolate the static design, the current reconfigurable module must be removed.

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 command to update the design with the blackbox and write the checkpoint.

```
source lock_placement_with_blackbox.tcl
```

The script will do the following tasks:

- Clear out the existing RMs executing the following commands.

  ```
  update_design -cell design_1_i/MAB1/inst/u1 -black_box

  update_design -cell design_1_i/MAB2/inst/u1 -black_box

  update_design -cell design_1_i/MAB3/inst/u1 -black_box

  update_design -cell
  design_1_i/OFDM/inst/DPR_QAM_QPSK_v1_0_S00_AXI_inst/T1/UUT/DAT_Mo
  d_Ins -black_box

  update_design -cell
  design_1_i/OFDM/inst/DPR_QAM_QPSK_v1_0_S00_AXI_inst/T1/UUT1/DataS
  ymDem_ins -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 Checkpoint/static_route_design.dcp
```

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

- Close the project.

```
Close_project
```

Create Other Configurations

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

5-1-1. Execute the following command to create and implement the second configuration, the UCB_T algorithm inside the MAB block with QPSK modulation scheme for the transceiver.

```
source create_second_configuration.tcl
```

The script will do the following tasks:

- First, it will open the blanking configuration using the tcl command:

```
open_checkpoint Checkpoint/static_route_design
```

- With the locked static design open in memory, read in post-synthesis checkpoint for the other reconfigurable modules.

```
read_checkpoint -cell design_1_i/MAB1/inst/u1
Synth/UCB_T/ucb_synth.dcp
```

```
read_checkpoint -cell design_1_i/MAB2/inst/u1
Synth/UCB_T/ucb_synth.dcp
```

```
read_checkpoint -cell design_1_i/MAB3/inst/u1
Synth/UCB_T/ucb_synth.dcp
```

```
read_checkpoint -cell
design_1_i/OFDM/inst/DPR_QAM_QPSK_v1_0_S00_AXI_inst/T1/UUT/DAT_Mo
d_Ins Synth/QPSK/mod_synth.dcp
```

```
read_checkpoint -cell
design_1_i/OFDM/inst/DPR_QAM_QPSK_v1_0_S00_AXI_inst/T1/UUT1/DataS
ymDem_ins Synth/QPSK/demod_synth.dcp
```

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

```
opt_design

place_design

route_design
```

- Save the full design checkpoint.

```
write_checkpoint -force
Implement/UCBT_QPSK/top_ucbt_qpsk_synth.dcp
```

- Close the project

```
close_project
```

Create Other Configurations

6-1. Read next set of RM dcp, create and implement the third configuration.

6-1-1. Execute the following command to create and implement the third configuration, the UCB_V algorithm inside the MAB block with QPSK modulation scheme for the transceiver.

6-1-2. `source create_third_configuration.tcl`

The script will do the following tasks:

- First, it will open the blanking configuration using the tcl command:

```
open_checkpoint Checkpoint/static_route_design
```

- With the locked static design open in memory, read in post-synthesis checkpoint for the second reconfigurable module.

```
read_checkpoint -cell design_1_i/MAB1/inst/u1
Synth/UCB_V/ucb_synth.dcp

read_checkpoint -cell design_1_i/MAB2/inst/u1
Synth/UCB_V/ucb_synth.dcp

read_checkpoint -cell design_1_i/MAB3/inst/u1
Synth/UCB_V/ucb_synth.dcp

read_checkpoint -cell
design_1_i/OFDM/inst/DPR_QAM_QPSK_v1_0_S00_AXI_inst/T1/UUT/DAT_Mo
d_Ins Synth/QPSK/mod_synth.dcp

read_checkpoint -cell
design_1_i/OFDM/inst/DPR_QAM_QPSK_v1_0_S00_AXI_inst/T1/UUT1/DataS
ymDem_ins Synth/QPSK/demod_synth.dcp
```

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

```
opt_design

place_design

route_design
```

- Save the full design checkpoint.

```
write_checkpoint -force Implement/UCBV_QPSK/top_route_design.dcp
```

- Close the project

```
close_project
```

Create the blanking configuration.

7-1-1. Execute the following command to create and implement the second configuration

```
source create_blanking_configuration.tcl
```

The script will do the following tasks:

- Open the static route checkpoint.

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

- 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.

  ```
  update_design -buffer_ports -cell design_1_i/MAB1/inst/u1

  update_design -buffer_ports -cell design_1_i/MAB2/inst/u1

  update_design -buffer_ports -cell design_1_i/MAB3/inst/u1

  update_design -buffer_ports -cell
  design_1_i/OFDM/inst/DPR_QAM_QPSK_v1_0_S00_AXI_inst/T1/UUT/DAT_Mo
  d_Ins

  update_design -buffer_ports -cell
  design_1_i/OFDM/inst/DPR_QAM_QPSK_v1_0_S00_AXI_inst/T1/UUT1/DataS
  ymDem_ins
  ```

- 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 Implement/BLANK/top_ucb_synth.dcp
  ```

- Close the project

  ```
  Close_project
  ```

Generate Bit Files

8-1. After all the Configurations have been validated by PR_Verify, full and partial bit files must be generated for the entire project

8-1-1. Generate the full configurations and partial bitstreams by executing the following tcl script.

```
source generate_bitstreams.tcl
```

8-1-2. The script will do the following tasks:

- Read the first configuration in the memory, using the command:

  ```
  open_checkpoint Implement/UCB_QAM/top_ucb_qam_synth.dcp
  ```

- Generate the full and partial bitstreams for this design.

  ```
  write_bitstream -file Bitstreams/UCB_QAM.bit

  close_project
  ```

- Read the blanking configuration in the memory, using the command:

  ```
  open_checkpoint Implement/BLANK/top_ucb_synth.dcp
  ```

- Generate a full bitstream with black boxes, plus blanking bitstreams for the reconfigurable modules. Blanking bitstreams can be used to “erase” an existing configuration to reduce power consumption.

  ```
  write_bitstream -file Bitstreams/BLANK.bit

  close_project
  ```

- Read the second configuration in the memory, using the command:

  ```
  open_checkpoint Implement/UCBT_QPSK/top_ucbt_qpsk_synth.dcp
  ```

- Generate full and partial bitstreams for the second configuration.

  ```
  write_bitstream -file Bitstreams/TUNED_QPSK.bit

  close_project
  ```

- Read the third configuration in the memory, using the command:

  ```
  open_checkpoint Implement/UCBV_QPSK/top_ucbt_qpsk_synth.dcp
  ```

- Generate full and partial bitstreams for the second configuration.

  ```
  write_bitstream -file Bitstreams/VAR_QPSK.bit

  close_project
  ```

Generate the Software Application — Step 10

9-1. Open the PS design that was created in Step 1. Export the hardware design and launch SDK.

9-1-1. Click on the **Open Project** link, browse to *c:/Summer/Tutorial/tutorial*, select the *tutorial.xpr* and click **OK** to open the design created in Step 1.

9-1-2. Select **File > Export > Export Hardware…**

9-1-3. In the *Export Hardware* form, do not check the *Include bitstream* checkbox and click **OK**.

9-1-4. Select **File > Launch SDK**

9-1-5. Click **OK** to launch SDK.

The SDK program will open. Close the Welcome tab if it opens.

9-2. Create a Board Support Package enabling generic FAT file system library.

9-2-1. In **SDK**, select **File** > **New** > **Board Support Package.**

9-2-2. Click **Finish** with the default settings (with standalone operating system).

This will open the Software Platform Settings form showing the OS and libraries selections.

9-2-3. Select **xilffs** as the FAT file support is necessary to read the partial bit files.

	Name	Version	Description
☐	libmetal	1.0	Libmetal Library
☐	lwip141	1.6	lwIP TCP/IP Stack library: lwIP v1.4.1
☐	openamp	1.1	OpenAmp Library
☑	xilffs	3.4	Generic Fat File System Library
☐	xilflash	4.2	Xilinx Flash library for Intel/AMD CFI com...
☐	xilisf	5.7	Xilinx In-system and Serial Flash Library
☐	xilmfs	2.1	Xilinx Memory File System
☐	xilpm	2.0	Power Management API Library for Zynq...
☐	xilrsa	1.2	Xilinx RSA Library
☐	xilskey	6.0	Xilinx Secure Key Library

Figure 13. Selecting the xilffs library support

9-2-4. Click **OK** to accept the settings and create the BSP.

9-3. Create an application.

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

9-3-2. Enter **TestApp** as the *Project Name*, and for *Board Support Package*, choose **Use Existing** (*standalone_bsp_0* should be the only option).

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

9-3-4. Expand the **TestApp** entry in the project view, right-click the *src* folder, and select **Import.**

9-3-5. Expand **General** category and double-click on **File System.**

9-3-6. Browse to *c:\Summer\Tutorial\Sources* and click **OK.**

9-3-7. Select **TestApp.c** and click **Finish** to add the file to the project.

9-3-8. Right-click on **TestApp** and select **C/C++ Building Settings.**

9-4. Create a zynq_fsbl application.

9-4-1. Select **File** > **New** > **Application Project.**

9-4-2. Enter **zynq_fsbl** as the *Project Name*, and for *Board Support Package*, choose **Create New**.

9-4-3. Click **Next**, select *Zynq FSBL,* and click **Finish.**

This will create the first stage bootloader application called zynq_fsbl.elf

9-5. Create a Zynq boot image.

9-5-1. Select **Xilinx Tools** > **Create Boot Image.**

9-5-2. Click the Browse button of the Output BIF file path field, browse to ***c:\Summer\Tutorial,*** and then click **Save** with the *output* as the default filename.

9-5-3. Click on the **Add** button of the *Boot image partitions,* click the Browse button in the Add Partition form, browse to ***c:\Summer\Tutorial*** **\tutorial\tutorial.sdk\zynq_fsbl\Debug** directory, select *zynq_fsbl.elf* and click **Open**.

9-5-4. 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:\Summer\Tutorial*** **\Bitstreams** directory, select *BLANK.bit* and click **Open**.

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

9-5-6. Click again on the **Add** button of the *Boot Image partitions,* click the Browse button in the Add Partition form, browse to ***c:\Summer\Tutorial*** **\tutorial\tutorial.sdk\TestApp\Debug** directory, select *TestApp.elf* and click **Open**.

9-5-7. Click **OK**.

9-5-8. Make sure that the output path is ***c:\Summer\Tutorial*** and the filename is *BOOT.bin,* and click **Create Image**.

9-5-9. 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.

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 HyperTerminal.

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. Follow the menu and test various reconfigurations.

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