Custom Peripheral for the AXI4-Full Interface

OBJECTIVES

- Create custom VHDL peripherals with an AXI4-Full Interface.
- Integrate a VHDL peripheral in a Block Based Design in Vivado.
- Create a software application in SDK that can handle the custom peripheral.
- Test the designs (code available for download) in a ZYBO board.

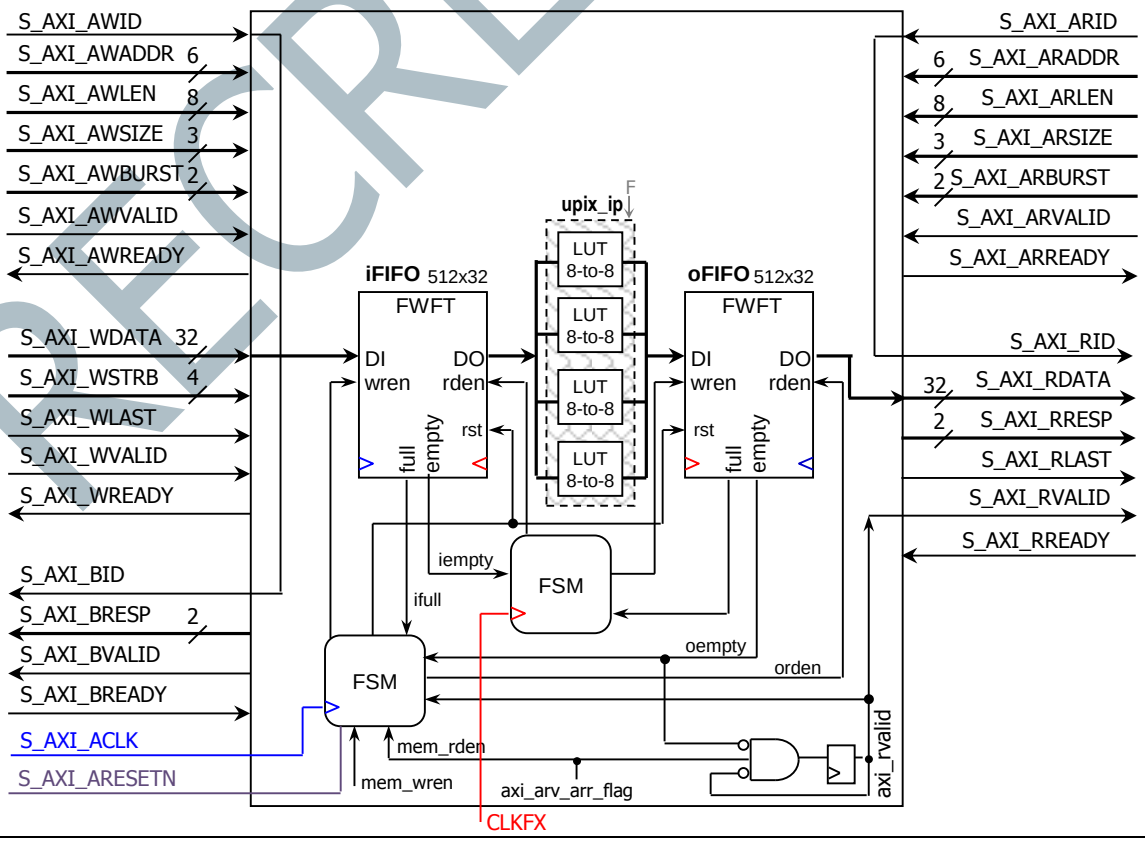
ZYBO BOARD SETUP FOR HARDWARE/SOFTWARE CO-DESIGN

- The current Zynq Book Tutorials (Aug 15th, 2015) includes a procedure that requires copying definition files into the Vivado installation directory. This helps when setting up the ZYBO Board.
- In this tutorial, we will manually indicate the Zynq device and the Processing System (PS) definition file.

PIXEL PROCESSOR: CUSTOM PERIPHERAL FOR AXI4-FULL INTERFACE

CONSIDERATIONS

- We will use the Pixel Processor with $NC = 4, NI = NO = 8$.
- As for the AXI4-Full Peripheral, the VHDL files allow for 3 cases:
  - ✓ MEMO: Xilinx® example just to test a simple 16-word (32 bits) memory architecture (no pixel processor).
  - ✓ PIXO: Similar to "MEMO", but we included the pixel processor (purely combinatorial) and a register between the data input and the memory. As in "MEMO", we can address 16 words.
  - ✓ **FIFO**: This is the custom FIFO-based interface that we will be using. All writes/read to any of the 16-word memory positions is treated the same (writing/reading on the FIFO).
- List of files to use:
  - ✓ `mypixfull_v1_0.vhd`: AXI4-Full Peripheral (top file).
  - ✓ `mypixfull_v1_0_S00_AXI.vhd`: AXI4-Full Interface description (the "**FIFO**" option is selected by default)
  - ✓ `myAXI_IP.vhd, my_AXI_fifo.vhd, my_AXImem.vhd, my_gen_pulse_sclr.vhd`: Ancillary files for the AXI4-Full Peripheral.
  - ✓ `static_ip.vhd`: top file for the Pixel Processor IP. Here, we can modify the parameter F (1..5).
  - ✓ `LUT_group.vhd, LUT_NItoNO.vhd, LUTNIto1.vhd, pack_xtras.vhd`.
  - ✓ `LUT_values8to8.txt`: LUT values.
- Pixel Processor AXI4-Full Peripheral (FIFO mode) with AXI signals (we make `S_AXI_CLK=CLK_FX`).



IP GENERATION

- Create new Vivado project. Select the **ZYNQ XC7Z010-1CLG400** device.
- Select Default Language VHDL. This way, the system wrapper and the template files for the AXI4-Full peripherals are created in VHDL.
- From the menu bar, select **Tools → Create and Package IP**. A new Vivado project will open.
  - ✓ Create a new AXI4 Peripheral. Name: mypixfull. Location `/my_repo`.
    Peripheral Repositories tip: To add a previously-generated IP into a new project, go to: Project Settings → IP → IP repositories and point to the repository folder. This is important if we created the repository in a folder when working on different project.
  - ✓ Add Interface: Full, 32 bits. Interface Mode: Slave. Memory Size: 64 bytes.
  - ✓ Select Edit IP. A New project appears, open it and look for the `<peripheral name>_S00_AXI.vhd` file (in this case it will be `mypixfull_v1_0_S00_AXI.vhd`) and **modify it** (i.e., replace it by our own file `mypixfull_v1_0_S00_AXI.vhd`). Same for the file `mypixfull_v1_0.vhd` (this is not strictly necessary though). Also, add all the other files (vhdl and ancillary files) to the folder `mypixfull_1.0/hdl`.
  - ✓ There is no need to add ports as our peripheral does not include external I/Os.
  - ✓ Synthesize (just to double-check everything is ok): You should've simulated the code in a different project.
  - ✓ Go to Package IP: Identify areas that need refresh: Click on 'Merge changes from File Group Wizard'.
  - ✓ Go to Review and Package → Edit packaging settings: Check 'create archive of IP', 'Close IP Packager Window', 'Add IP to the IP Catalog in the current project' (do not check 'Delete project after packaging'). Then click on Re-Package IP.
- You will return to the original Vivado Project.

CREATING A BLOCK DESIGN PROJECT IN VIVADO

- Click on 'Create Block Design' and instantiate the Zynq PS and the new peripheral (Select Add IP, look for our brand new IP and add it.).
- Click on 'Run Block Automation' and 'Run Connection Automation'.
- Double click on ZYNQ_PS: Load the `ZYBO_zynq_def.xml` file (Import XPS settings). This indicates which peripherals are used by the PS. If the file is not loaded, the software application on the ddr or a peripheral (e.g. UART) will not work properly.
- We do not need to add the .xdc file as our peripheral does not use external ports.
- Create the VHDL wrapper (Sources Window → right click on the top-level system design)
- Synthesize, implement, and generate the bitstream.
  - ✓ It will not work at first. But the `/mypixfull_v1_0/` folder will be created in the project folder (`.srcs/sources_1/…/ipshared/xilinx.com/`). We need to place any ancillary files (e.g. .txt files) in the `/hdl` portion of that folder to make it work.
- Export hardware (with bitstream) and launch SDK.

SDK

- See Tutorial Unit 2 for instructions on how to create a software application on SDK.
- Go to Xilinx Tools → Repositories, click on 'New' and then browse to the folder `\my_repo\mypixfull_1.0` and click ok.
- Create a new SDK application. Then, copy the following file into the `/src` folder: `test_pixi.c`
- The idea is to control the AXI4-Full peripheral (Pixel Processor) with software instructions by writing and reading from memory positions. Note that the instructions to write on the AXI4-Full peripheral are different than in the case of AXI4-Lite.
- Make sure the computer has the FTDI drivers installed (for UART). We might also need to configure the UART to 115200 (this is how it is configured in the PS), also we might need to configure SDK terminal with the same parameters (115200)
- **Testing strategy**: write four 32-bit words and read four 32-bit words:
  - ✓ `MYPIXFULL_mWriteMemory(Base Address, 32-bit word)`: Use this function (created when generating the AXI-4 Full IP) to write a 32-bit word onto the AXI4-Full Peripheral.
  - ✓ `MYPIXFULL_mReadMemory(Base Address)`: Use this function (created when generating the AXI4-Full IP) to read a 32-bit word from the AXI4-Full Peripheral.
  - ✓ `Base Address`: In the FIFO case, any address in the 64 byte range (the one allowed for the AXI4-Full Pixel Processor peripheral) works since we only write/read to/from the FIFOs.
  - ✓ For example, with the given Pixel Processor Parameters (with parameter F = 1), we get:

Input	Output
0xDEADBEEF	0xEED2DDF7
0xBEBEDEAD	0xDDDDEED2
0xFADEBEAD	0xFDEEDDD2
0xCAFEBEDF	0xE3FFDDEF

2D DCT (DISCRETE COSINE TRANSFORM): CUSTOM PERIPHERAL FOR AXI4-FULL INTERFACE

CONSIDERATIONS

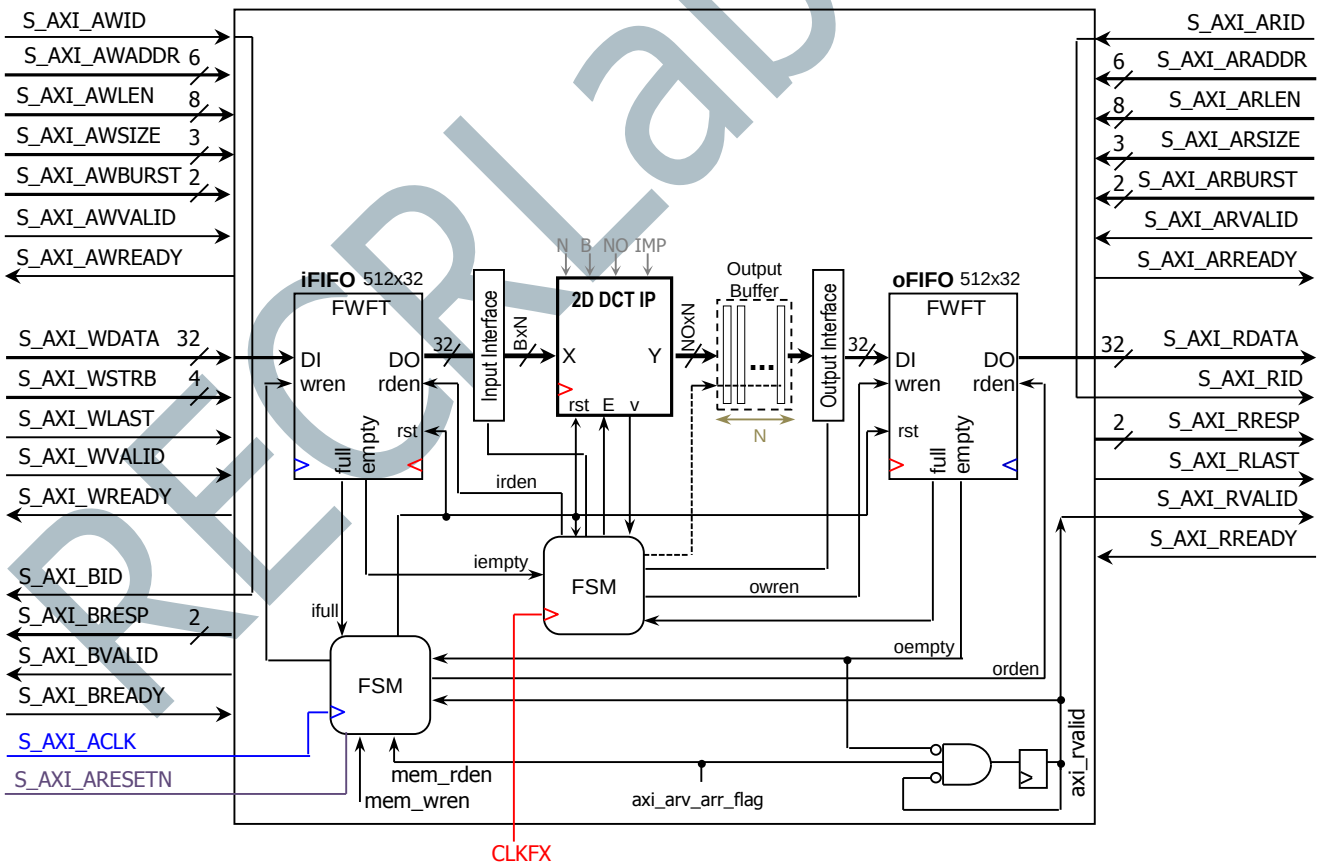
- We will use the 2D DCT hardware that allows for transforms of sizes 4, 8, and 16.
- As for the AXI4-Full Peripheral, the VHDL files allows only for the case:
  - ✓ **FIFO**: This is the custom FIFO-based interface that we will be using. All writes/read to any of the 16-word memory positions is treated the same (writing/reading on the FIFO).

- Description of the files (the most important ones) we are using:
  - ✓ `mydctfull_v1_0.vhd`: AXI4-Full Peripheral (top file).
  - ✓ `mydctfull_v1_0_S00_AXI.vhd`: AXI4-Full Interface description (the "**FIFO**" option is the only available)
  - ✓ `myAXI_IP.vhd, my_AXI_fifo.vhd, my_gen_pulse_sclr.vhd`: Ancillary files for the AXI4-Full Peripheral.
  - ✓ `dct_ip.vhd`: top file of the hardware that runs at CLKFX. This includes the 2D DCT IP, the input/output interfaces to the FIFOs, and the Output Buffer.
  - ✓ `DCT_2d.vhd`: Hardware architecture for the 2D DCT.
  - ✓ `fsm_fullypip.vhd, fsm_onetrans.vhd`: Hardware description of the FSM @ CLKFX.
  - ✓ `DCT4_NH16_LUT_values[1..4], DCTeo8_NH16_LUT_values[1..8], DCTeo16_NH16LUT_values[1..16].txt`: LUT values for the DCT Transform coefficients.

- 2D DCT parameters: In `my_AXI_fifo.vhd`, we can modify: N (transform size: 4, 8, 16), B (input pixel bitwidth: 8, 16), NO (output pixel bitwidth: 8,16), IMP (implementation type: folded, pipelined), NH (coefficient bit-width). We fix NH=16.

- You can use these files as a template to build a AXI4-Full peripheral for any architecture. You would only to modify the file `my_AXI_fifo.vhd`: instead of instantiating `dct_ip`, you need to instantiate the component containing the new architecture, its respective glue logic to the FIFOs, and the FSM @ CLKFX.

- 2D DCT AXI4-Full Peripheral (FIFO-based) with AXI signals (we make `S_AXI_CLK=CLK_FX`).



IP GENERATION

- Create new Vivado project. Select the **ZYNQ XC7Z010-1CLG400** device.
- Select Default Language VHDL. This way, the system wrapper and the template files for the AXI4-Full peripherals are created in VHDL.
- From the menu bar, select **Tools → Create and Package IP**. A new Vivado project will open.
  - ✓ Create a new AXI4 Peripheral. Name: mypixfull. Location `/my_repo`.
    Peripheral Repositories tip: To add a previously-generated IP into a new project, go to: Project Settings → IP → IP repositories and point to the repository folder. This is important if we created the repository in a folder when working on different project.
  - ✓ Add Interface: Full, 32 bits. Interface Mode: Slave. Memory Size: 64 bytes.
  - ✓ Select Edit IP. A New project appears, open it and look for the `<peripheral name>_S00_AXI.vhd` file (in this case it will be `mydctfull_v1_0_S00_AXI.vhd`) and **modify it** (i.e., replace it by our own file `mydctfull_v1_0_S00_AXI.vhd`). Same for the file `mydctfull_v1_0.vhd` (this is not strictly necessary though). Also, add all the other files (vhdl and ancillary text files) to the folder `mydctfull_1.0/hdl`.
  - ✓ There is no need to add ports as our peripheral does not include external I/Os.
  - ✓ Synthesize (just to double-check everything is ok): You should've simulated the code in a different project.
  - ✓ Go to Package IP: Identify areas that need refresh: Click on 'Merge changes from File Group Wizard'.
  - ✓ Go to Review and Package → Edit packaging settings: Check 'create archive of IP', 'Close IP Packager Window', 'Add IP to the IP Catalog in the current project' (do not check 'Delete project after packaging'). Then click on Re-Package IP.
- You will return to the original Vivado Project.

CREATING A BLOCK DESIGN PROJECT IN VIVADO

- Click on 'Create Block Design' and instantiate the Zynq PS and the new peripheral (Select Add IP, look for our brand new IP and add it.).
- Click on 'Run Block Automation' and 'Run Connection Automation'.
- Double click on ZYNQ_PS: Load the `ZYBO_zynq_def.xml` file (Import XPS settings). This indicates which peripherals are used by the PS. If the file is not loaded, the software application on the ddr or a peripheral (e.g. UART) will not work properly.
- We do not need to add the .xdc file as our peripheral does not use external ports.
- Create the VHDL wrapper (Sources Window → right click on the top-level system design)
- Synthesize, implement, and generate the bitstream.
  - ✓ It will not work at first. But the `/mydctfull_v1_0` folder will be created in `.srcs/sources_1/…/ipshared/xilinx.com/`. We need to place any ancillary files (e.g. .txt) in the `/hdl` folder to make it work.
- Export hardware (with bitstream) and launch SDK.

SDK

- See Tutorial Unit 2 for instructions on how to create a software application on SDK.
- Go to Xilinx Tools → Repositories, click on 'New' and then browse to the folder `\my_repo\mydctfull_1.0` and click ok.
- Create a new SDK application. Then, copy the following file into the `/src` folder: `dct_tst.c`
- The idea is to control the AXI4-Full 2D DCT peripheral with software instructions by writing and reading from memory positions. Note that the instructions to write on the AXI4-Full peripheral are different than in the case of AXI4-Lite.

- **Testing strategy**: With $N = 4,\ NO = 16,\ B = 8$. For every 2D Transform, we write 4 input columns (four 32-bit words), and we get 4 output columns (eight 32-bit words).
  - ✓ `MYDCTFULL_mWriteMemory(Base Address, 32-bit word)`: Use this function (created when generating the AXI-4 Full IP) to write a 32-bit word onto the AXI4-Full Peripheral.
  - ✓ `MYDCTFULL_mReadMemory(Base Address)`: Use this function (created when generating the AXI4-Full IP) to read a 32-bit word from the AXI4-Full Peripheral.
  - ✓ `Base Address`: In the FIFO case, any address in the 64 byte range (the one allowed for the AXI4-Full 2D DCT peripheral) works since we only write/read to/from the FIFOs.
  - ✓ With the given 2D DCT parameters, we provide two data sets. For the input values, each 32-bit word is a column; for the output values, each two 32-bit words is a row.

Input (columns)	Output (rows)
0xDEADBEEF	0x8000E92E
0xBEBEDEAD	0x14C00D82
0xFADEBEAD	0x18A6E418
0xCAFEBEDF	0xDB3E1FB2
	0x0A401E19
	0x1D40236D
	0xF8382A32
	0xDEC9FDE7

Input (columns)	Output (rows)
0xCFC7C9C7	0x80000CF4
0xCAC4C6C3	0xFF0003D5
0xC6C3C7C3	0x0471045F
0xBEBDC2BD	0xFF89FF65
	0x010003CE
	0x0000000B
	0x06D0FFE5
	0x00310020