Custom Peripheral for the AXI4-Lite Interface

OBJECTIVES

- Create a Hardware/Software System using the ZYBO Board or the ZYBO Z7-10 Board.
- Create custom VHDL peripherals with an AXI4-Lite Interface.
- Integrate a VHDL peripheral in a Block Based Design in Vivado 2019.1.
- Create a software application in SDK that can transfer data from/to the custom peripheral.

ZYBO/ZYBO Z7-10 BOARD SETUP FOR HARDWARE/SOFTWARE CO-DESIGN

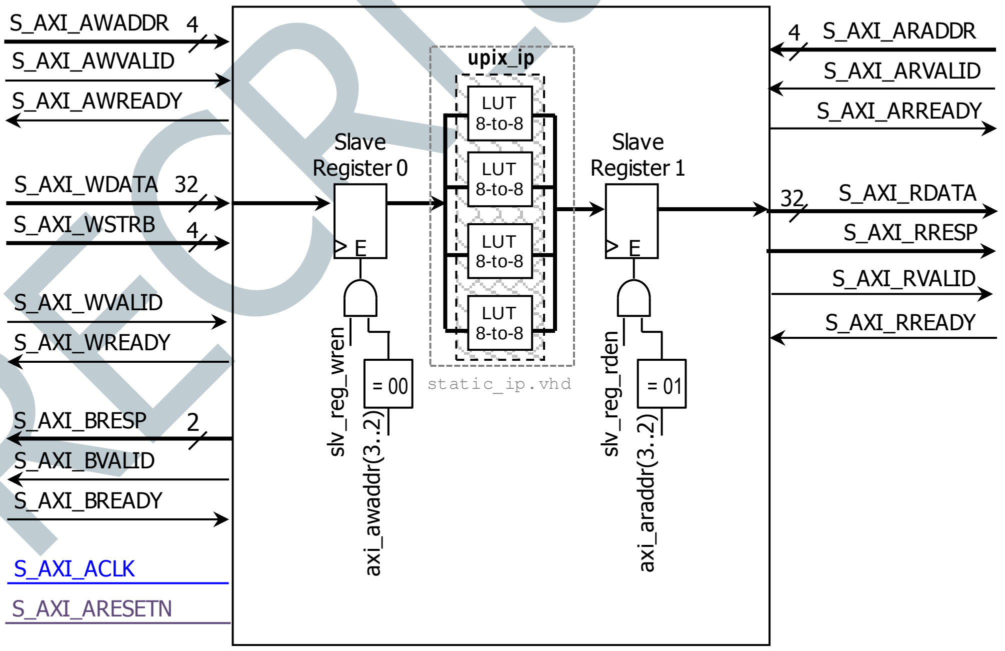
- It is assumed that the definition files (available in `vivado-boards-mastes.zip`) have been installed in Vivado.
- ZYBO: PS_CLK input: 50 MHz. PL_CLK input: 125 MHz. By default, a 100 MHz is generated for the PL fabric.
- ZYBO Z7-10: PS_CLK input: 33.33 MHz. PL_CLK input: 125 MHz. By default, a 50 MHz clock is generated for the PL fabric.
- Refer to the Zynq Book Tutorial: IP Creation → Creating IP in VHDL for detailed step-by-step instructions on how to integrate a custom hardware in Vivado.

PIXEL PROCESSOR: CUSTOM PERIPHERAL FOR AXI4-LITE INTERFACE

CONSIDERATIONS

- We will use the Pixel Processor with $NC = 4, NI = NO = 8, F = 1$.
- List of files to use:
  - ✓ `mypix_v1_0.vhd`: AXI4-Lite peripheral (top file, Vivado template)
  - ✓ `mypix_v1_0_S00_AXI.vhd`: AXI4-Lite interface description (edited Vivado template)
  - ✓ `static_ip.vhd`: Pixel Processor IP with connection to the Slave Registers.
  - ✓ `LUT_group.vhd`: Top file of the Pixel Processor IP
  - ✓ `LUT_NItoNO.vhd, LUT_NIto1.vhd, pack_xtras.vhd`: Other files that make up the Pixel Processor.
  - ✓ `LUT_values8to8.txt`: LUT values in a text file.
  - ✓ `tb_mypixAXI4Lite.vhd`: Testbench for AXI4-Lite peripheral. This is very useful as it emulates the AXI signals resulting from the execution of the software in the PS. This allows us to fix peripheral errors.

- We need two Slave Registers to process data through this Pixel Processor circuit (one for writing data, one for reading data).



IP GENERATION

- Create a new project in Vivado: `myaxilitepix`.
  - ✓ Make sure the default language is VHDL, so that the system wrapper and template files are created in VHDL
  - ✓ At Default Part, go to Boards, and select the **Zybo** (or **Zybo Z7-10**) Board.
- From the menu bar, select **Tools** → **Create and Package New IP**.

- ✓ Create a new AXI4 Peripheral. Name: `mypix`. Location `/ip_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 associated repository folder.
- ✓ Add Interface: Lite, 32 bits, 4 registers (we just need 2, but 4 is the minimum).
- ✓ 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 `mypix_v1_0_S00_AXI.vhd`). Modify the project by:
  - Using only the required registers, i.e., commenting out VHDL code that specifies unused registers and instantiating (port-map) the pixel processor IP in the file `mypix_v1_0_S00_AXI.vhd`. As a shortcut, you can just replace this file with the file `mypix_v1_0_S00_AXI.vhd` file that is available for download.
  - Adding the extra files to the folder `/hdl` in `/ip_repo/mypix_1.0` and adding these source files (including the .txt file) to the Vivado project. * Vivado 2019.1: by default, the files will be added to the folder `/src`.
- ✓ There is no need to add ports as our peripheral does not include external I/Os.
- ✓ Synthesize your circuit (just to double-check everything is ok): You should've simulated this code in a different project.
- ✓ The following instructions are detailed in the Zynq Book Tutorial: IP → Creating IP in VHDL (Return to IP Packager, Review and Package):
  - Go to Package IP - mypix: Identify areas that need refresh. In this project, we only added files, so click on File Groups. Then 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 (don't check Delete project after Packaging). Then, click on Re-Package IP.

- Your custom IP is now ready to be used as an AXI4-Lite Peripheral.
- 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 AXI MYPIX peripheral.
- Click on Run Block Automation and Run Connection Automation. Then 'Validate Design'
- There is no need to add an `.xdc` file as our peripheral does not use external ports.
- Create the VHDL wrapper (Sources Window → right click on the top-level system design → Create HDL Wrapper).
- Synthesize, implement, and generate the bitstream.
  - ✓ An error will be reported when Synthesizing. Vivado only copies VHDL files from the IP folder to the embedded project folder (located inside the `/<peripheral name>.srcs/…/ipshared` folder). As a result, the `LUT_NItoNO.vhd` file cannot find the `LUT_values.txt`. We need to place this text file in the same folder as the `LUT_NItoNO.vhd` file.
  - ✓ This folder location is available by opening the `LUT_NItoNO.vhd` file. You need to find this file in the design structure or via the Vivado error which will point to the `LUT_NItoNO.vhd` file. After copying the .txt file, you can Synthesize again.
  - ✓ In general, this procedure is to be followed for any ancillary file (e.g. text file) used by the VHDL files.
- Export hardware (with bitstream) and launch SDK

SOFTWARE APPLICATION IN SDK

- Use Tutorial Unit 2 for instructions on how to create and test a software application on SDK.
- Navigate to Xilinx Tools → Repositories, click on 'New' and then browse to the folder `\ip_repo\mypix_1.0` and click ok.
- Create a new SDK application: pixtest. Then, copy the following file into the `/src` folder: `pixproc_test.c`. This file will test all the possible inputs to each 8-bit LUT (`0x00` to `0xFF`): The 32-bit input word will have four identical bytes. Example:
  - ✓ Input = `0x01010101`, Expected Result = `0x10101010`
  - ✓ Input = `0x03030303`, Expected Result = `0x1C1C1C1C`
  - ✓ Input = `0xFDFDFDFD`, Expected Result = `0xFEFEFEFE`

PIPELINED DIVIDER: CUSTOM PERIPHERAL FOR AXI4-LITE

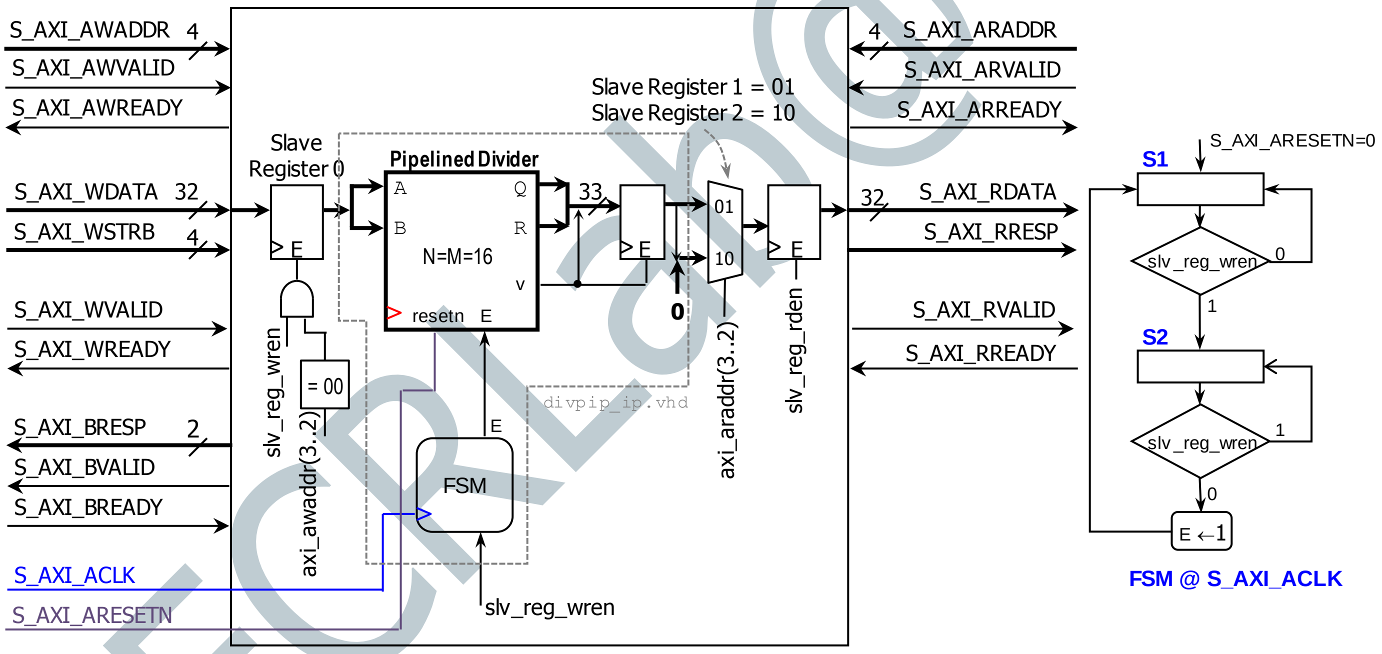
CONSIDERATIONS

- We will use the Pipelined Integer Divider with $N = 16, M = 16$.
- List of files to use:
  - ✓ `mydiv_v1_0.vhd`: AXI4-Lite peripheral (top file). This is the same file generated by Vivado.
  - ✓ `mydiv_v1_0_S00_AXI.vhd`: AXI4-Lite Interface description. This file is generated by Vivado, but it has been edited to include the Pipelined Divider IP (`divpip_ip.vhd`).
  - ✓ `divpip_ip.vhd`: Pipelined Divider with some interfacing (FSM) to the Slave Registers for AXI4-Lite Interfacing.
  - ✓ `res_div_pip.vhd`: Top file of the Pipelined Divider IP.
  - ✓ `fulladd.vhd, my_pashiftreg.vhd, unit_proc.vhd, dffe.vhd`: Other files that make up the Pipelined Divider.
  - ✓ `tb_mydivAXI4Lite.vhd`: Testbench for AXI4-Lite peripheral. This is very useful as it emulates the AXI signals resulting from the execution of the software in the PS. This allows us to fix peripheral errors.
- We need 3 Slave Registers to process data through this Pipelined Divider circuit (one for writing data, two for reading data).

IP GENERATION

- Create a new project in Vivado: `myaxilitediv`.
  - ✓ Make sure the default language is VHDL, so that the system wrapper is created in VHDL
  - ✓ At Default Part, go to Boards, and select the **Zybo** (or **Zybo Z7-10**) Board.
- From the menu bar, select **Tools → Create and Package New IP**.
  - ✓ Create a new AXI4 Peripheral. Name: `mydiv`. Location `/ip_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 associated repository folder.
  - ✓ Add Interface: Lite, 32 bits, 4 registers (we just need 3, but 4 is the minimum).
  - ✓ 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 `mydiv_v1_0_S00_AXI.vhd`). Modify the project by:
    - ▫ Using only the required registers, i.e., commenting out VHDL code that specifies unused registers and instantiating (port-map) the pipelined divider IP in the file `mydiv_v1_0_S00_AXI.vhd`. As a shortcut, you can just replace this file with the file `mydiv_v1_0_S00_AXI.vhd` file that is available for download.
    - ▫ Adding the extra files to the folder `/hdl` in `/ip_repo/mydiv_1.0` and adding these source files to the Vivado project.
      * Vivado 2019.1: by default, the files will be added to the folder `/src`.
  - ✓ There is no need to add ports as our peripheral does not include external I/Os.
  - ✓ Synthesize your circuit (just to double-check everything is ok): You should've simulated this code in a different project.
  - ✓ Go to Package IP – mydiv -→ File Groups (Merge changes). Then Review and Package → Re-Package IP.
- Your custom IP is now ready to be used as an AXI4-Lite Peripheral.
- 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 AXI MYDIV peripheral.
- Click on Run Block Automation and Run Connection Automation. Then 'Validate Design'
- There is no need to add an `.xdc` file as our peripheral does not use external ports.
- Create the VHDL wrapper (Sources Window → right click on the top-level system design → Create HDL Wrapper).
- Synthesize, implement, and generate the bitstream.
- Export hardware (with bitstream) and launch SDK

SOFTWARE APPLICATION IN SDK

- Navigate to Xilinx Tools → Repositories, click on 'New' and then browse to the folder `\ip_repo\mydiv_1.0` and click ok.
- Create a new SDK application: divtest. Then, copy the following file into the `/src` folder: `div_test.c`. This file will test three integer divisions:
  - ✓ `A = 0x008C, B = 0x0009`. Expected Result: `Q = 0x000F, R = 0x0005`.
  - ✓ `A = 0x00BB, B = 0x000A`. Expected Result: `Q = 0x0012, R = 0x0007`.
  - ✓ `A = 0x0FEA, B = 0x0371`. Expected Result: `Q = 0x0004, R = 0x0226`.

PIPELINED 2D CONVOLUTION KERNEL: CUSTOM PERIPHERAL FOR AXI4-LITE

CONSIDERATIONS

- We will use the Pipelined 2D Convolution Kernel with $B = C = 8, N = 3, REP = "UNSIGNED"$.
- List of files to use:
  - ✓ `myconv2_v1_0.vhd`: AXI4-Lite Peripheral (top file). This is the same file generated by Vivado.
  - ✓ `myconv2_v1_0_S00_AXI.vhd`: AXI4-Lite Interface description. This file is generated by Vivado, but it has been edited to include the 2D Convolution Kernel IP.
  - ✓ `myconv2_ip.vhd`: 2D Convolution Kernel IP with some interfacing (FSM) to the Slave Registers for AXI4-Lite Interfacing. In addition, here the input matrix `H` of the 2D Convolution Kernel is fixed to:

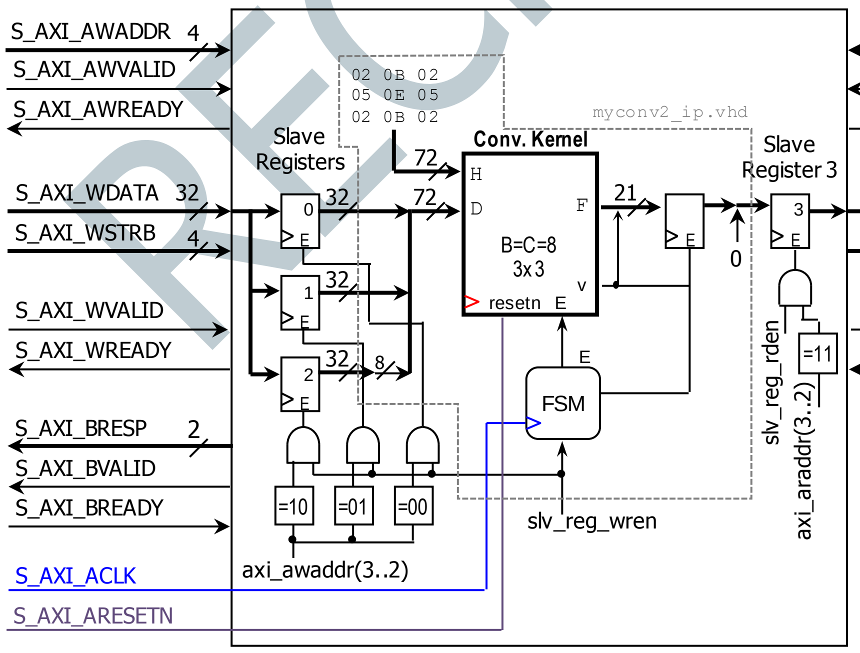
$$\begin{bmatrix} 0x02 & 0x0B & 0x02 \\ 0x05 & 0x0E & 0x05 \\ 0x02 & 0x0B & 0x02 \end{bmatrix}$$

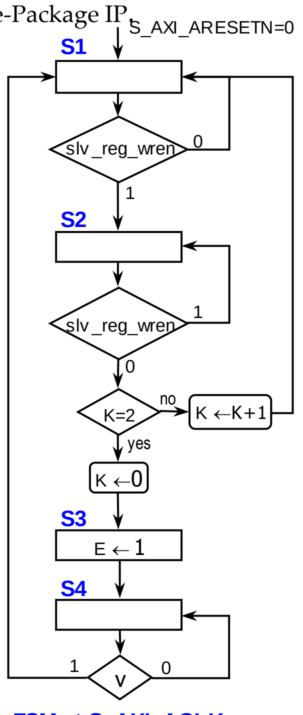
  - ✓ `myconv2.vhd`: Top file of the 2D Convolution Kernel IP.
  - ✓ `adder_tree.vhd`, `my_pashiftreg.vhd`, `my_rege.vhd`, `my_addsub.vhd`, `fulladd.vhd`, `dffe.vhd`, `pack_xtras.vhd`: Files that make up the Pipelined 2D Convolution Kernel.
  - ✓ `tb_myconv2_lite.vhd`: Testbench for AXI4-Lite peripheral. This is very useful as it emulates the AXI signals resulting from the execution of the software in the PS. This allows us to fix peripheral errors.
- The Pipelined 2D Convolution Kernel requires 4 Slave Registers for processing data (3 for writing data, 1 for reading data).

IP GENERATION

- Create a new project in Vivado: `myaxiliteconv2`.
  - ✓ Make sure the default language is VHDL, so that the system wrapper is created in VHDL
  - ✓ At Default Part, go to Boards, and select the **Zybo** (or **Zybo Z7-10**) Board.
- From the menu bar, select **Tools** → **Create and Package New IP**.
  - ✓ Create a new AXI4 Peripheral. Name: `myconv2`. Location `/ip_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 associated repository folder.
  - ✓ Add Interface: Lite, 32 bits, 4 registers.
  - ✓ 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 `myconv2_v1_0_S00_AXI.vhd`). Modify the project by:
    - ▫ Using only the required registers, i.e., commenting out VHDL code that specifies unused registers and instantiating (port-map) the pixel processor IP in the file `myconv2_v1_0_S00_AXI.vhd`. As a shortcut, you can just replace this file with the file `myconv2_v1_0_S00_AXI.vhd` file that is available for download.
    - ▫ Adding the extra files to the folder `/hdl` in `/ip_repo/myconv2_1.0` and adding these source files to the Vivado project. * Vivado 2019.1: by default, the files will be added to the folder `/src`.
  - ✓ There is no need to add ports as our peripheral does not include external I/Os.
  - ✓ Synthesize your circuit (just to double-check everything is ok): You should've simulated this code in a different project.
  - ✓ Go to Package IP – myconv2 → File Groups (Merge changes). Then Review and Package → Re-Package IP.
- You will return to the original Vivado Project.





FSM at S_AXI_ACLK

CREATING A BLOCK DESIGN PROJECT IN VIVADO

- Click on Create Block Design and instantiate the Zynq PS and the AXI MYCONV2 peripheral.
- Click on Run Block Automation and Run Connection Automation.
- There is no need to add an `.xdc` file as our peripheral does not use external ports.
- Create the VHDL wrapper (Sources Window → right click on the top-level system design → Create HDL Wrapper).
- Synthesize, implement, and generate the bitstream.
- Export hardware (with bitstream) and launch SDK

SOFTWARE APPLICATION IN SDK

- Navigate to Xilinx Tools → Repositories, click on New and then browse to the folder `\ip_repo\myconv2_1.0` and click ok.
- Create a new SDK application: conv2test. Then, copy the following file into the `/src` folder: `myconv2_test.c`. This file will test three input cases:
  - ✓
    ```
    D = [A1 B2 C3
         D4 F0 E1
         D2 C3 B3].
    ```
    Expected Result: `0x00102B82`
  - ✓
    ```
    D = [F1 09 05
         0A C3 02
         A1 F0 1C].
    ```
    Expected Result: `0x001018FF`
- Note that the application also measures elapsed time (us) between input data is written and output data is retrieved.