Using Direct Memory Access (DMA)

OBJECTIVES

- Transfer data between memory regions as well as between memory and a custom peripheral with the DMA Controller inside the PS (SDK 2019.1).
- Use the AXI4-Full Pixel Processor Peripheral (from Unit 4 or Unit 7) to test DMA transfers.
- Learn to develop software routines for DMA Transfers with interrupts and test on ZYBO or ZYBO Z7-10 Board.

DMA CONTROLLER - DOCUMENTATION

- UG585: Zynq-7000 AP SoC Technical Reference Manual.
- SDK: Go to `system.mss` → `Peripheral Drivers` → `ps7_dma_s` → `Import Examples`. You can import the file `xdmaps_example_w_intr.c`. The software routine in this tutorial is based on this file.

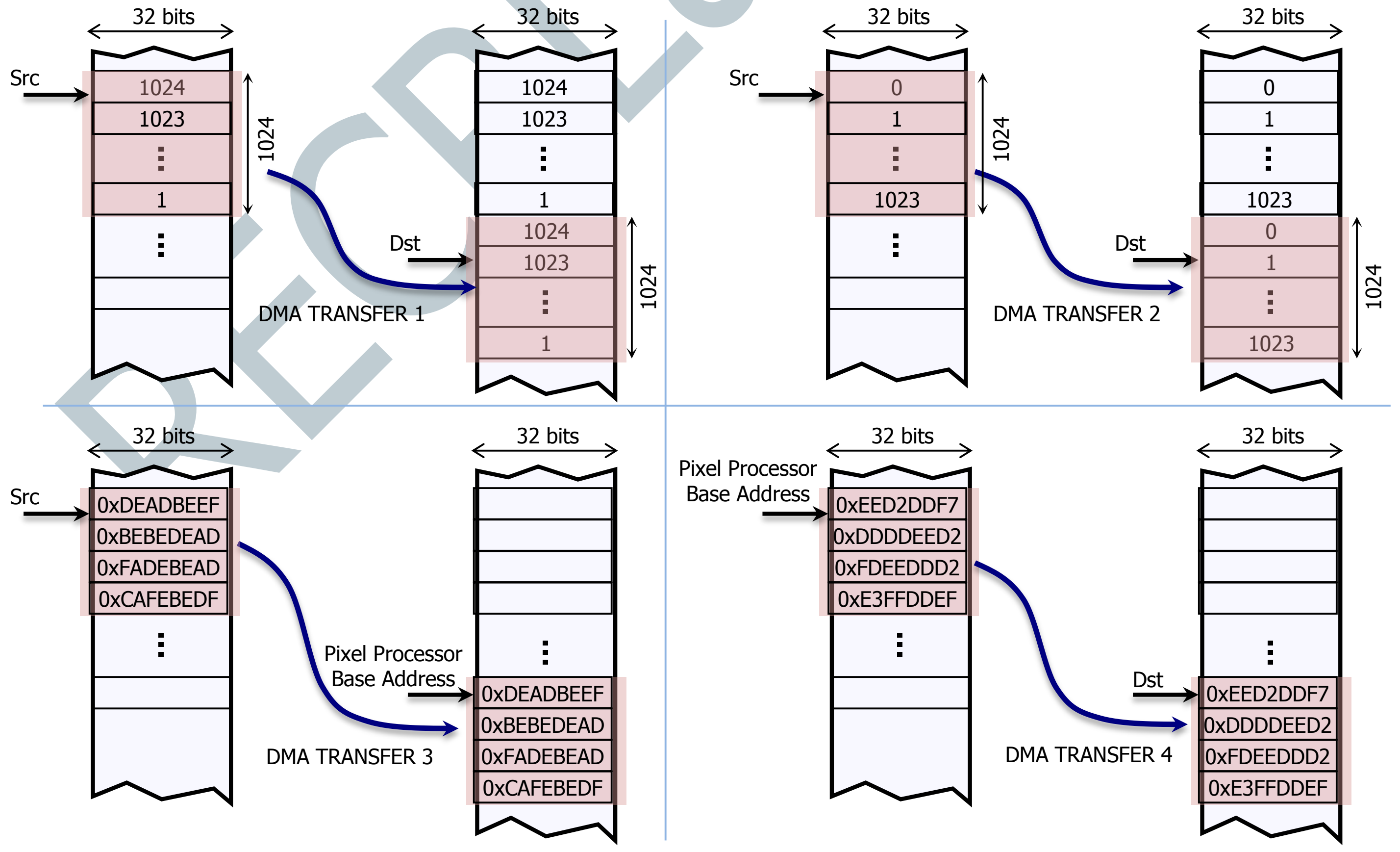
DMA TEST

- The test project is the AXI-4 Full Pixel Processor peripheral (Unit 4 or Unit 7). If using the example of Unit 7, open the Vivado embedded system and use the associated bitstream (the one not created by the Partial Reconfiguration flow).
- **Pixel Processor:** The circuit, written in VHDL, processes NC NI-bit pixels in parallel and outputs NC NO-bit pixels. We set `NC=4`, `NI=NO=8` in this test. Also, we use the default parameter `F=1`.
- The following four transfers are carried out, one after the other. The DMA Length is in 32-bit words.

Transfer type	**DMA Length**	**Channel**	**Source Data**	**Notes**
Memory to Memory	1024	0	1024 down to 1	Any Channel can be used.
Memory to Memory	1024	0	0 to 1023	Channel 0 is re-used by first making it idle.

Memory to AXI-4 Full Peripheral	4	1	`0xDEADBEEF`
			`0xBEBEDEAD`
			`0xFADEBEAD`
			`0xCAFEBEDF`
				Channel 1. We write data onto the Pixel Processor.

AXI-4 Full Peripheral to Memory	4	2	`0xEED2DDF7`
			`0xDDDDEED2`
			`0xFDEEDDD2`
			`0xE3FFDDEF`
				Channel 2. We retrieve data from the Pixel Processor.



- For a list of available commands, see the file `xdmaps.h` in the *bsp*: `/libsrc/dmaps_v2_1/src`.
- We need to specify a variable of type `XDmaPs_Cmd`. We then clear all the fields of the variable and fill them up with information such as Source Address, Destination Address, DMA Length.
- To start a DMA Transfer, we use the command `XDmaPs_Start`.
- To indicate the completion of a DMA transaction, the DMAC issues interrupts: DMA Done Interrupt (0-7) and DMA Fault Interrupt. Each Interrupt has its associated Interrupt Service Routine (ISR). For a tutorial on Interrupts, see Unit 9.
  - ✓ We first must connect the ISRs to the Generic Interrupt Controller (via `XScuGic_Connect`).
  - ✓ We can execute a user-defined function inside an ISR. This needs to be specified before a DMA Transaction (`XDmaPs_Start`) by using the function `XdmaPs_SetDoneHandler`, where we indicate the DMA instance, the channel being used (0-7), our callback function (e.g. `DmaDoneHandler`) and our callback reference data (e.g.: `Checked[8]`).
  - ✓ In the `DmaDoneHandler` function, we can specify the instructions we want to be executed once an interrupt hits.

- For ease of explanation, we provide two examples:
  - ✓ After each DMA transaction, we wait a certain amount of time deemed sufficient for the DMA transfers to finish (e.g.: printing via UART). This is a simple and inefficient method, but it is helpful to introduce DMA.
  - ✓ We use the DMA interrupt to detect the exact moment the DMA transactions ends. Every DMA channel (0-7) can issue a 'done' interrupt. This method is more efficient and it is the preferred method, though it requires significantly more lines of code to set up and configure the interrupts.

PROCEDURE – NO INTERRUPTS (USING DELAY)

- Open the Vivado project of the AXI-4 Full Pixel Processor peripheral (Unit 4 or Unit 7).
- Open the SDK Project of the AXI-4 Full Pixel Processor peripheral.
- Create a new SDK application.
  - ✓ Go to New → Application Project. On Project Name, you can use: `dma_test`.
  - ✓ In Board Support Package (*bsp*): You can create a new one or use a previously generated one.
- Copy the following files in the `/src` folder: `pix_dma.c`.
- Go to File → Generate Linker Script. If required, make sure to assign enough space in the heap/stack for the data. Also, place the code/heap/stack section in DDR memory (the largest one).
- Once the program is compiled, connect the ZYBO (or ZYBO Z7-10) Board to the USB port of your computer.
- Download the bitstream on the PL: Xilinx Tools → Program FPGA.
- Go to SDK Terminal and connect to the proper COM port.
- Select the project `dma_test`. Right click and select Run As → Launch on Hardware (GDB).
- Verification:
  - ✓ DMA Transfer 1: The program prints out destination data: `Dst[0]` to `Dst[1023]`. It should match the Source data.
  - ✓ DMA Transfer 2: The program prints out destination data: `Dst[0]` to `Dst[1023]`. It should match the Source data.
  - ✓ DMA Transfer 3: Data is written into the Pixel Processor IP. This will be verified in DMA Transfer 4.
  - ✓ DMA Transfer 4: Data is retrieved from Pixel Processor IP. The program prints out destination data: `Dst[0]` to `Dst[3]`. It should match the output data from Pixel Processor (with `F=1`):

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

PROCEDURE – USING DMA CHANNEL INTERRUPTS

- Open the Vivado project of the AXI-4 Full Pixel Processor peripheral (Unit 4 or Unit 7).
- Open the SDK Project of the AXI-4 Full Pixel Processor peripheral.
- Create a new SDK application.
  - ✓ Go to New → Application Project. On Project Name, you can use: `dma_test_intr`.
  - ✓ In Board Support Package (*bsp*): You can create a new one or use a previously generated one.
- Copy the following files in the `/src` folder: `pix_dma_intr.c`.
- Go to File → Generate Linker Script. If required, make sure to assign enough space in the heap/stack for the data. Also, place the code/heap/stack section in DDR memory (the largest one).
- Once the program is compiled, connect the ZYBO Board to the USB port of your computer.
- Download the bitstream on the PL: Xilinx Tools → Program FPGA.
- Go to SDK Terminal and connect to the proper COM port.
- Select the project `dma_test_intr`. Right click and select Run As → Launch on Hardware (GDB).
- Verification: This is a similar procedure to the one without interrupts.
  - ✓ Before every transaction, we set `Checked[Channel] = 0` and link the callback function `DmaDoneHandler` and the callback reference data `Checked[8]` to the respective ISR via the function `XdmaPs_SetDoneHandler`.
  - ✓ After every transaction, we use function `wait_doneint (Checked, Channel)` to wait until the interrupt hits before proceeding on to next instructions.