DMA-controlled Accelerator for FFT (Lab 4)

Chester Sungchung Park (박성정)
SoC Design Lab, Konkuk University
Webpage: http://soclab.konkuk.ac.kr

Outline

- ❑ Objectives
- ❑ Lab 4: DMA-controlled accelerator

Objectives

❑ After completing this lab, you will be able to:

- Speed up accelerators by using DMA-based data transfer instead of PS-based data transfer

Lab 4: PS-Controlled Accelerator

❑ Creating IP projects

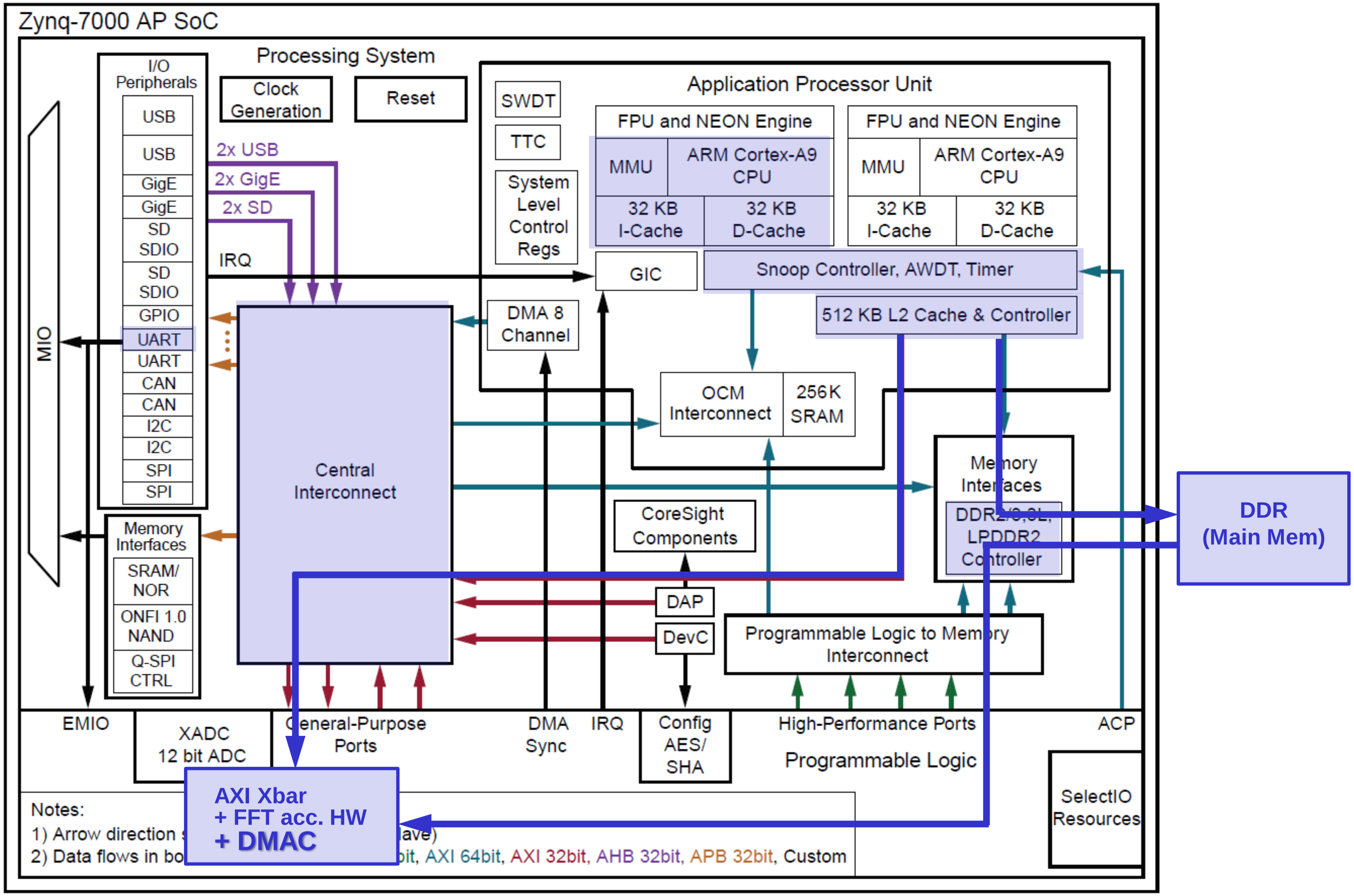
❑ Creating block designs

❑ Generating bitstream

❑ Running C applications

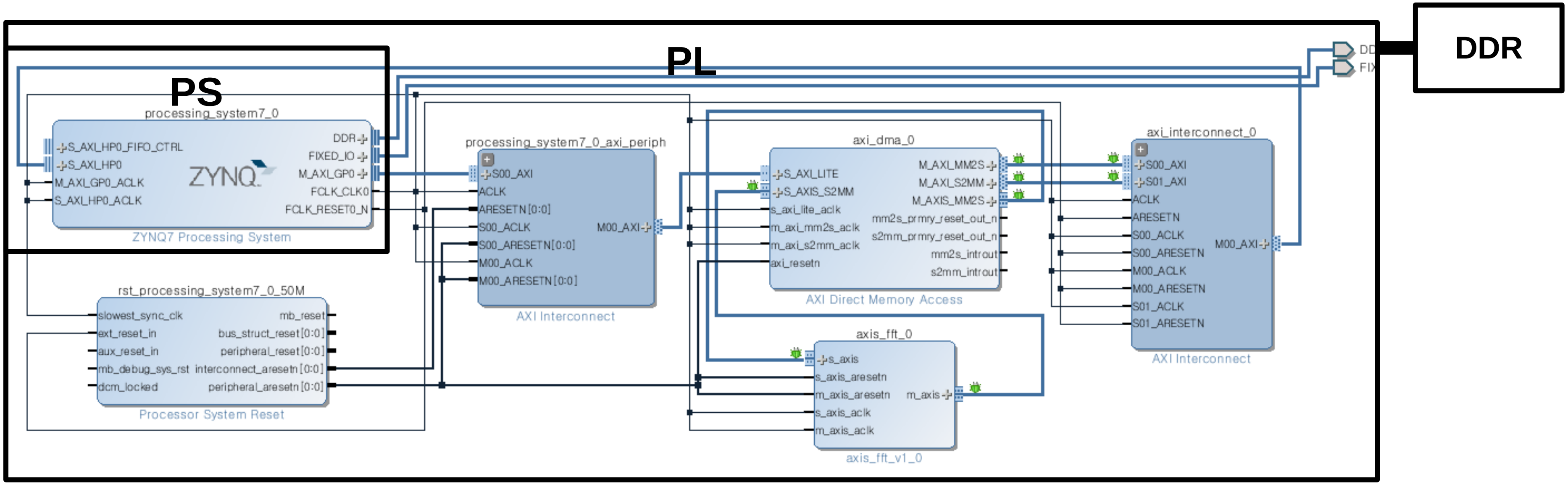
❑ Debugging designs in Integrated Logic Analyzer (ILA)

Block Diagram



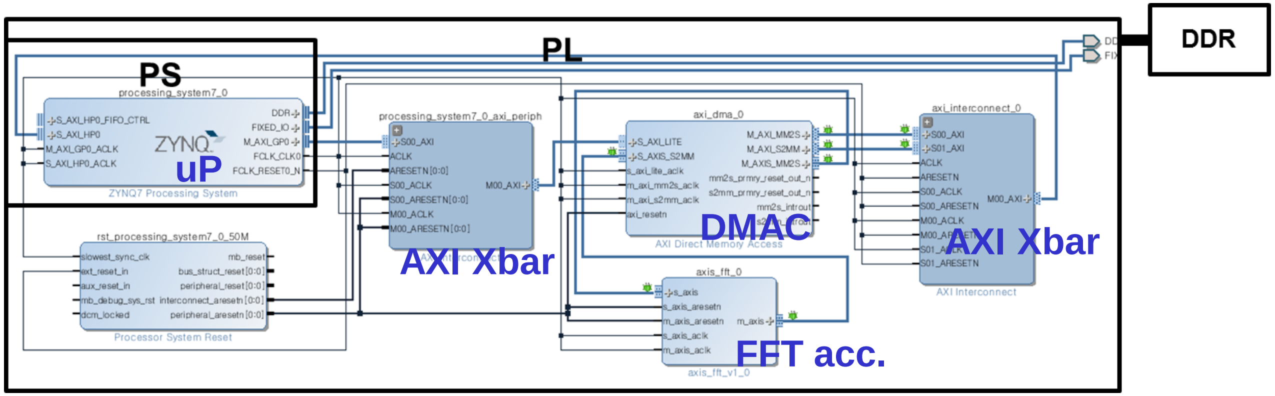
Block Diagram

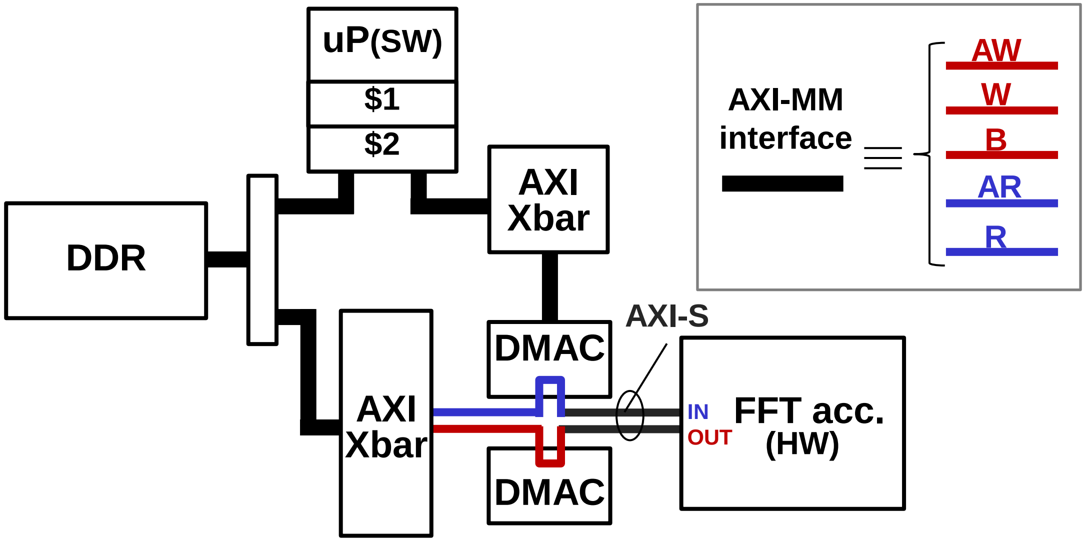
- ❑ SoC integration



Block Diagram

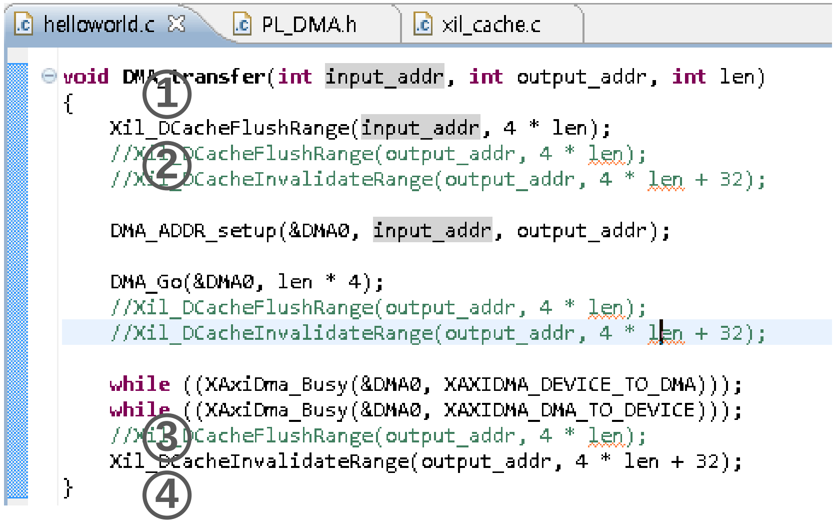
- ❑ SoC integration



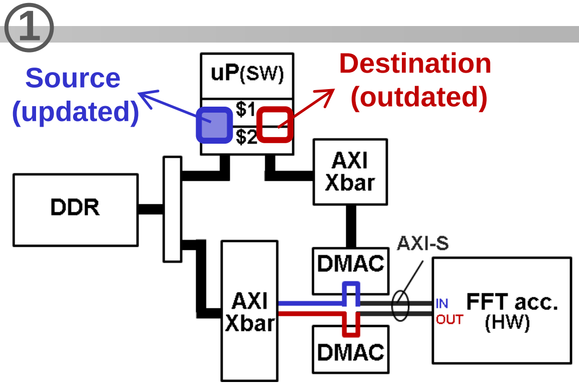


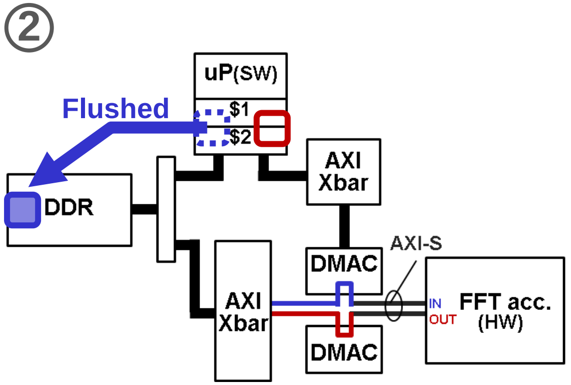
FFT Accelerator

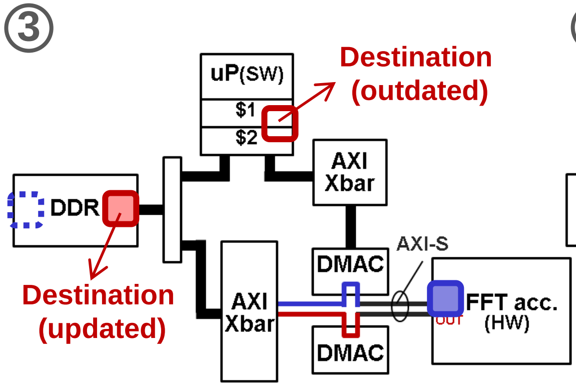
❑ Dataflow

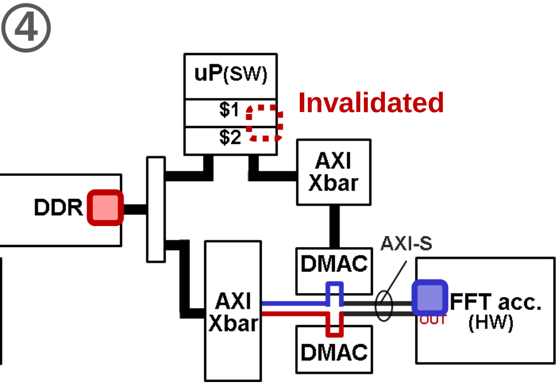


```
void DMA_transfer(int input_addr, int output_addr, int len)
{
    Xil_DCacheFlushRange(input_addr, 4 * len);
    //Xil_DCacheFlushRange(output_addr, 4 * len);
    //Xil_DCacheInvalidateRange(output_addr, 4 * len + 32);

    DMA_ADDR_setup(&DMA0, input_addr, output_addr);

    DMA_Go(&DMA0, len * 4);
    //Xil_DCacheFlushRange(output_addr, 4 * len);
    //Xil_DCacheInvalidateRange(output_addr, 4 * len + 32);

    while ((XAxiDma_Busy(&DMA0, XAXIDMA_DEVICE_TO_DMA)));
    while ((XAxiDma_Busy(&DMA0, XAXIDMA_DMA_TO_DEVICE)));
    //Xil_DCacheFlushRange(output_addr, 4 * len);
    Xil_DCacheInvalidateRange(output_addr, 4 * len + 32);
}
```



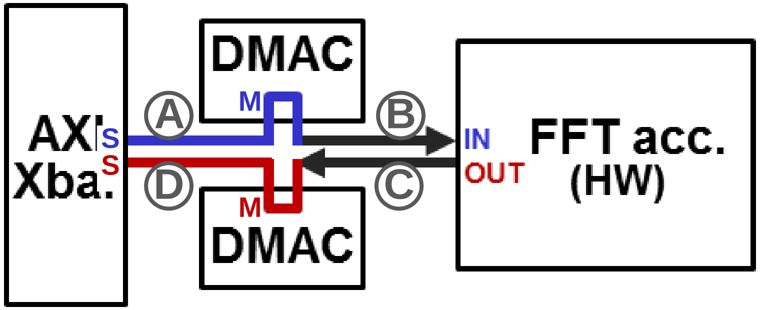


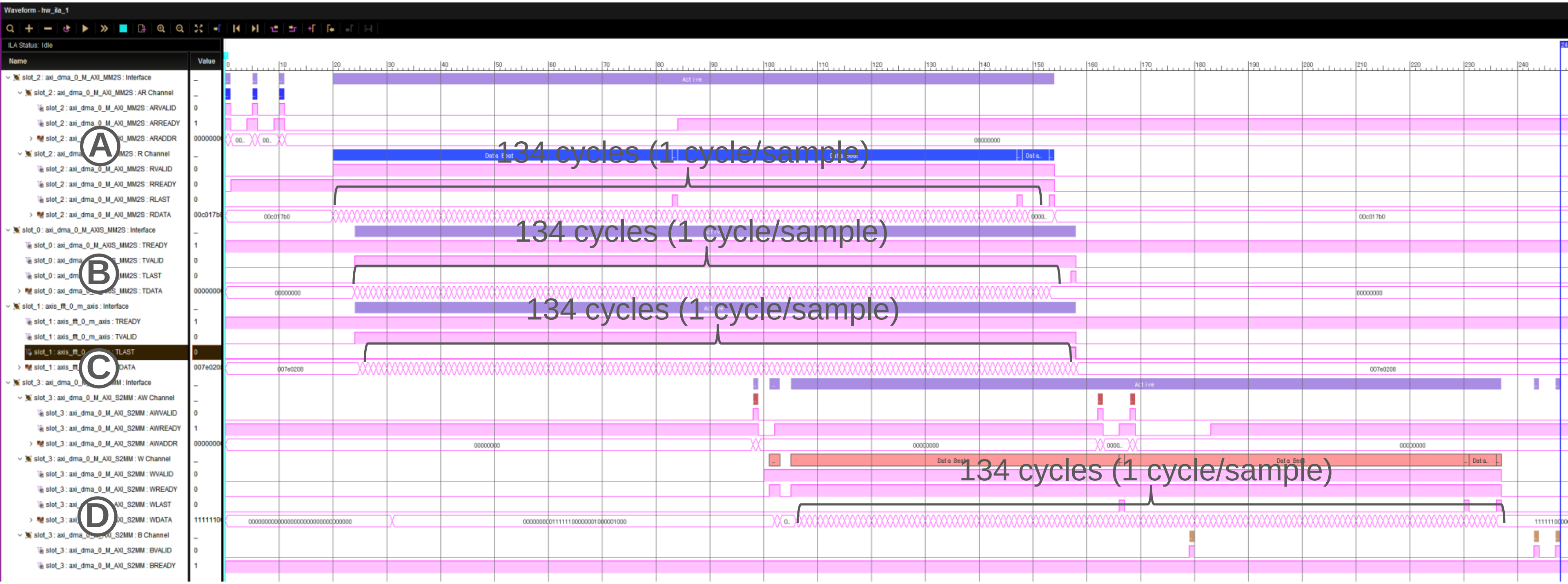




FFT Accelerator

- ❑ Timing diagram
  - Ⓐ: AXI-MM (AR,R)
  - ⒷⒸ: AXI-S
  - Ⓓ: AXI-MM (AW,W,B)

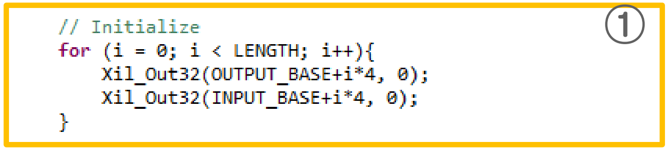



Waveform - hw_ila_1
ILA Status: Idle
Active
134 cycles (1 cycle/sample)
134 cycles (1 cycle/sample)
134 cycles (1 cycle/sample)
134 cycles (1 cycle/sample)
Ⓐ
Ⓑ
Ⓒ
Ⓓ

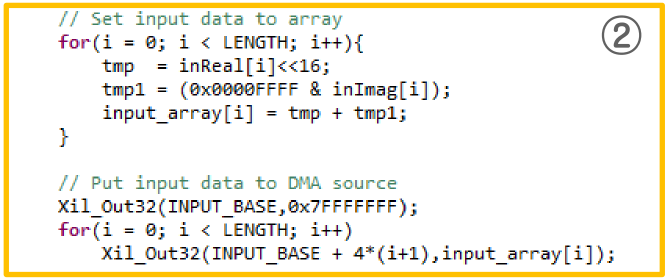
Name	Value
slot_2 : axi_dma_0_M_AXI_MM2S : Interface	_
slot_2 : axi_dma_0_M_AXI_MM2S : AR Channel	_
slot_2 : axi_dma_0_M_AXI_MM2S : ARVALID	0
slot_2 : axi_dma_0_M_AXI_MM2S : ARREADY	1
slot_2 : axi_ XI_MM2S : ARADDR	0000000
slot_2 : axi_dma M2S : R Channel	_
slot_2 : axi_dma_0_M_AXI_MM2S : RVALID	0
slot_2 : axi_dma_0_M_AXI_MM2S : RREADY	0
slot_2 : axi_dma_0_M_AXI_MM2S : RLAST	0
slot_2 : axi_dma_0_M_AXI_MM2S : RDATA	00c017b
slot_0 : axi_dma_0_M_AXIS_MM2S : Interface	_
slot_0 : axi_dma_0_M_AXIS_MM2S : TREADY	1
slot_0 : axi_dma S_MM2S : TVALID	0
slot_0 : axi_dm MM2S : TLAST	0
slot_0 : axi_dma_0_ IS_MM2S : TDATA	0000000
slot_1 : axis_fft_0_m_axis : Interface	_
slot_1 : axis_fft_0_m_axis : TREADY	1
slot_1 : axis_fft_0_m_axis : TVALID	0
slot_1 : axis_fft_0 : TLAST	0
slot_1 : axis_fft DATA	007e020
slot_3 : axi_dma_0 M : Interface	_
slot_3 : axi_dma_0_M_AXI_S2MM : AW Channel	_
slot_3 : axi_dma_0_M_AXI_S2MM : AWVALID	0
slot_3 : axi_dma_0_M_AXI_S2MM : AWREADY	1
slot_3 : axi_dma_0_M_AXI_S2MM : AWADDR	0000000
slot_3 : axi_dma_0_M_AXI_S2MM : W Channel	_
slot_3 : axi_dma_0_M_AXI_S2MM : WVALID	0
slot_3 : axi_dma_0_M_AXI_S2MM : WREADY	0
slot_3 : axi XI_S2MM : WLAST	0
slot_3 : axi XI_S2MM : WDATA	1111110
slot_3 : axi_dma XI_S2MM : B Channel	_
slot_3 : axi_dma_0_M_AXI_S2MM : BVALID	0
slot_3 : axi_dma_0_M_AXI_S2MM : BREADY	1

Source Codes

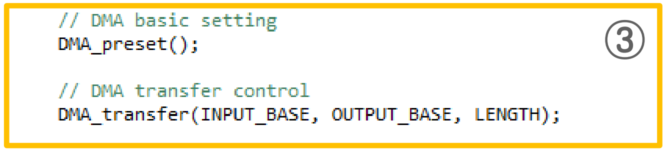
- ❑ main()
  - ① Initialize input/output base address
  - ② Assign input data into an array
  - ③ Set DMA control registers
  - ④ Assign output data into an array
  - ⑤ Convert output data to binary



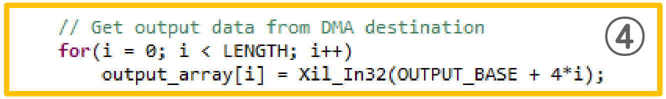
```
// Initialize
for (i = 0; i < LENGTH; i++){
    Xil_Out32(OUTPUT_BASE+i*4, 0);
    Xil_Out32(INPUT_BASE+i*4, 0);
}
```



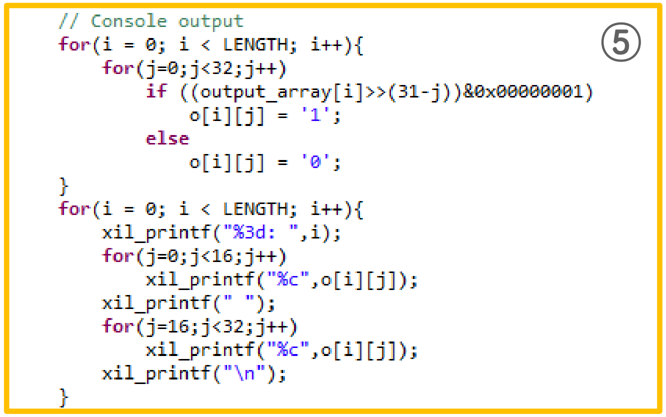
```
// Set input data to array
for(i = 0; i < LENGTH; i++){
    tmp  = inReal[i]<<16;
    tmp1 = (0x0000FFFF & inImag[i]);
    input_array[i] = tmp + tmp1;
}

// Put input data to DMA source
Xil_Out32(INPUT_BASE,0x7FFFFFFF);
for(i = 0; i < LENGTH; i++)
    Xil_Out32(INPUT_BASE + 4*(i+1),input_array[i]);
```



```
// DMA basic setting
DMA_preset();

// DMA transfer control
DMA_transfer(INPUT_BASE, OUTPUT_BASE, LENGTH);
```



```
// Get output data from DMA destination
for(i = 0; i < LENGTH; i++)
    output_array[i] = Xil_In32(OUTPUT_BASE + 4*i);
```



```
// Console output
for(i = 0; i < LENGTH; i++){
    for(j=0;j<32;j++)
        if ((output_array[i]>>(31-j))&0x00000001)
            o[i][j] = '1';
        else
            o[i][j] = '0';
}
for(i = 0; i < LENGTH; i++){
    xil_printf("%3d: ",i);
    for(j=0;j<16;j++)
        xil_printf("%c",o[i][j]);
    xil_printf(" ");
    for(j=16;j<32;j++)
        xil_printf("%c",o[i][j]);
    xil_printf("\n");
}
```

Source Codes

❑ DMA_transfer()

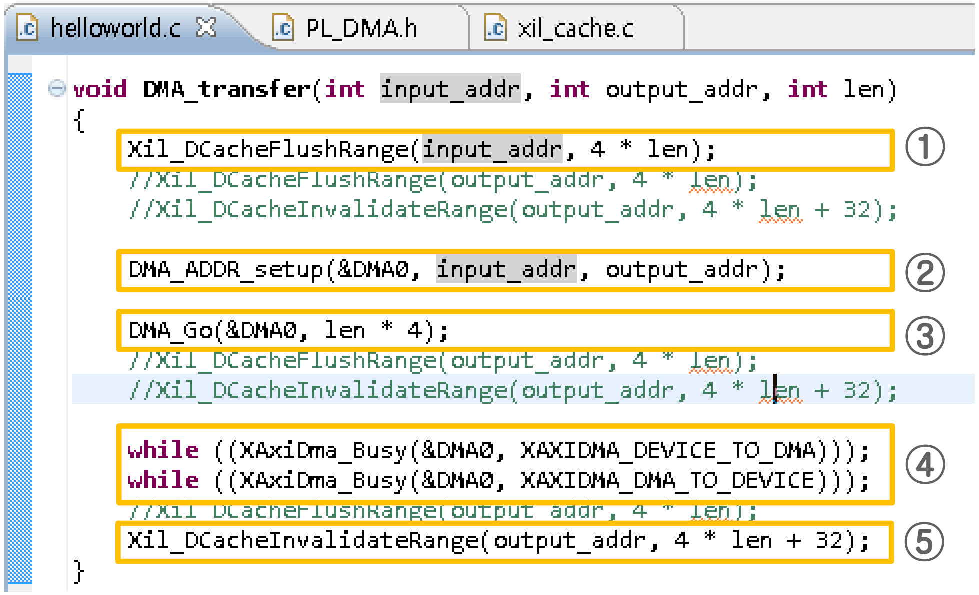
① Flush the source region of the D-cache

② Set the (start) addresses of the source & destination regions

③ Set the burst length (i.e., the number of beats per burst)

④ Check the DMA status register to see if the transfer ends

⑤ Invalidate the destination region of the D-cache



```c
void DMA_transfer(int input_addr, int output_addr, int len)
{
    Xil_DCacheFlushRange(input_addr, 4 * len);
    //Xil_DCacheFlushRange(output_addr, 4 * len);
    //Xil_DCacheInvalidateRange(output_addr, 4 * len + 32);

    DMA_ADDR_setup(&DMA0, input_addr, output_addr);

    DMA_Go(&DMA0, len * 4);
    //Xil_DCacheFlushRange(output_addr, 4 * len);
    //Xil_DCacheInvalidateRange(output_addr, 4 * len + 32);

    while ((XAxiDma_Busy(&DMA0, XAXIDMA_DEVICE_TO_DMA)));
    while ((XAxiDma_Busy(&DMA0, XAXIDMA_DMA_TO_DEVICE)));
    //Xil_DCacheFlushRange(output_addr, 4 * len);
    Xil_DCacheInvalidateRange(output_addr, 4 * len + 32);
}
```