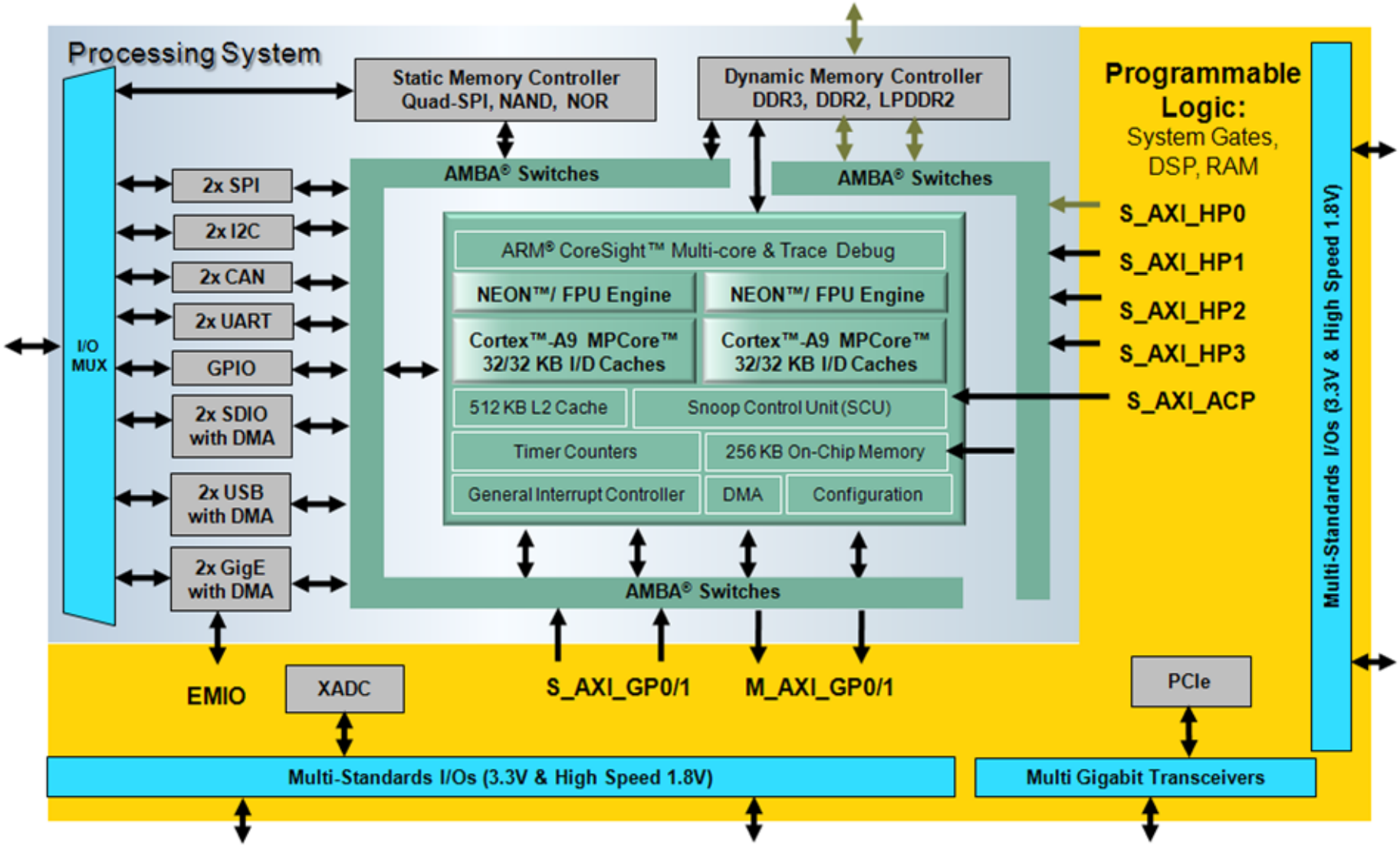
ELD Lab 9
FFT Using FPGA and ARM Processor

Objective

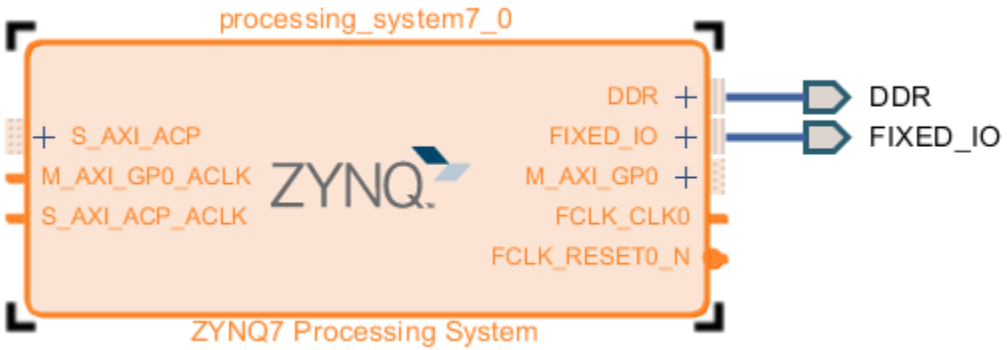
- Implement 8-point FFT on FPGA and ARM Cortex A9 processor of Zynq SoC and compare their execution time.
- **Homework 1:** Implement 16-point FFT on FPGA and ARM Cortex A9 processor of Zynq SoC and compare their execution time.

Theory & Lab

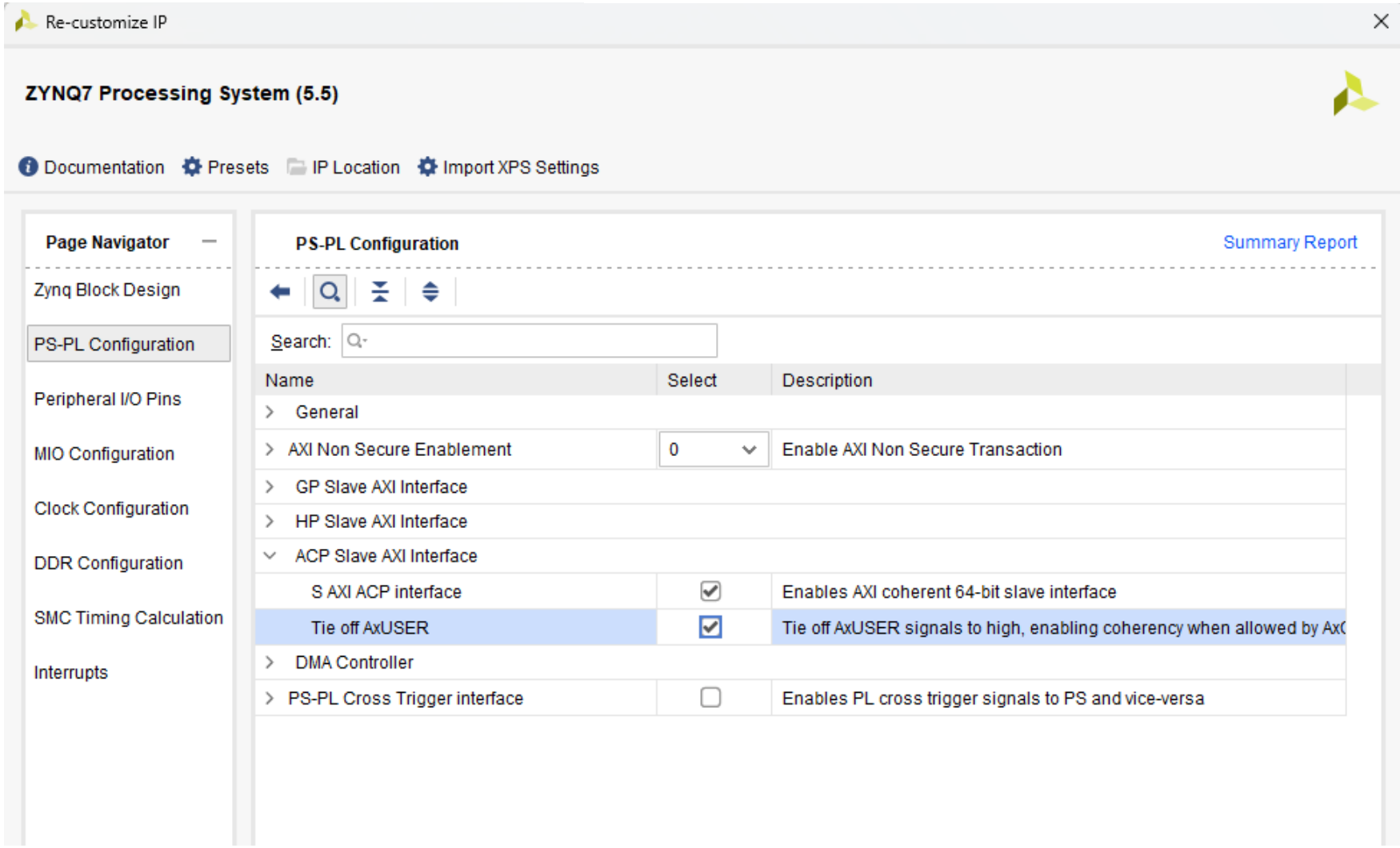
Zynq Architecture: PS and PL



Enable ACP and GP Ports

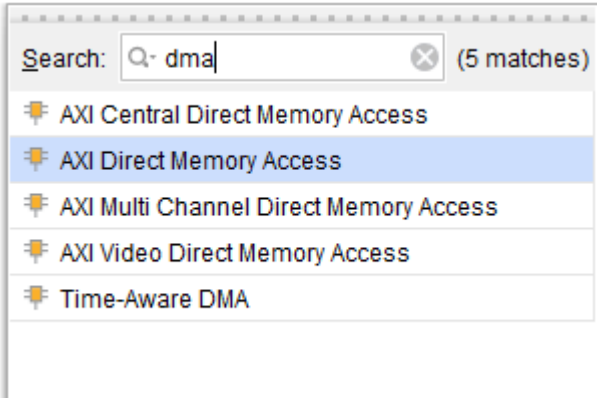


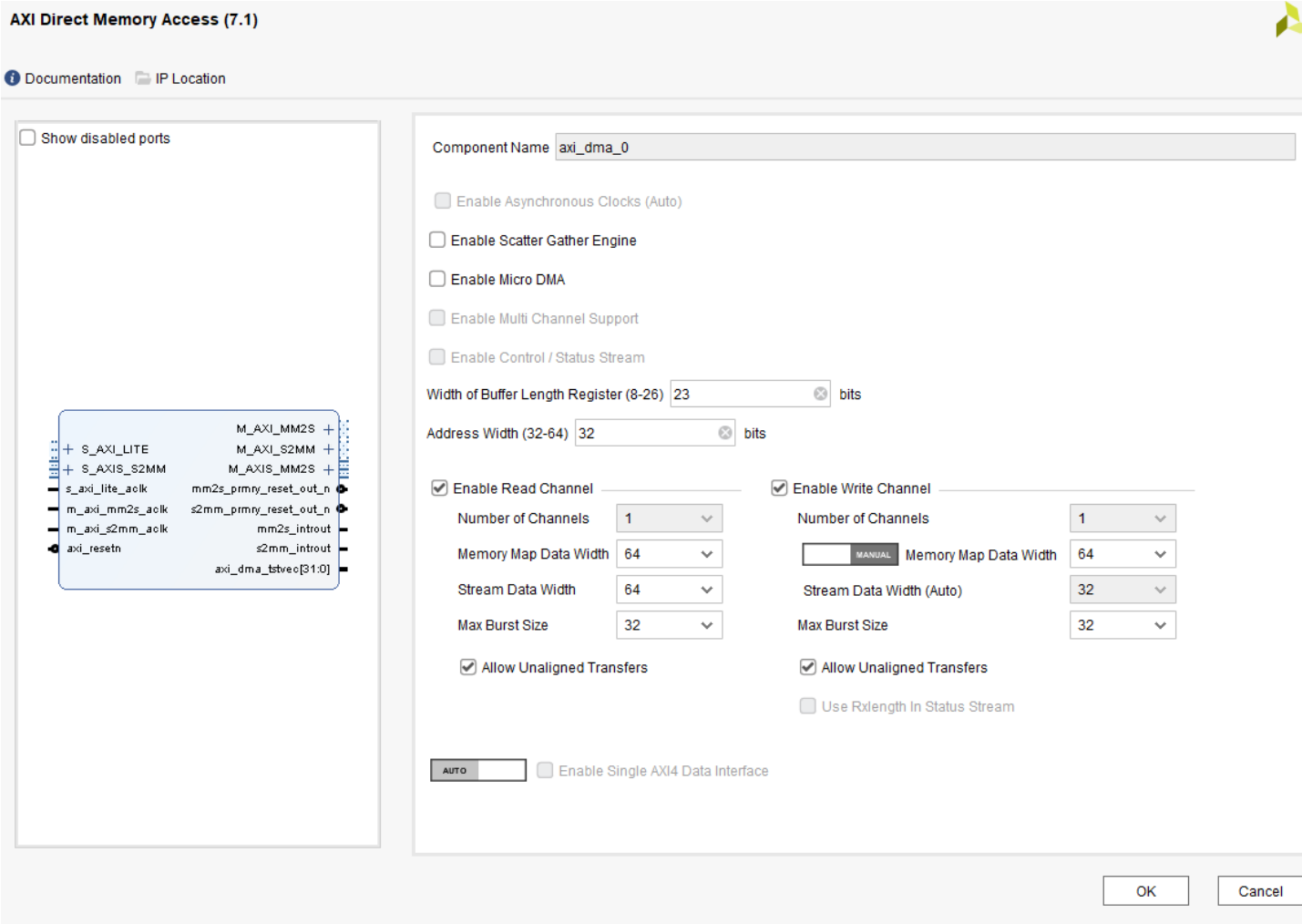
- Enable ACP and GP ports along with UART
- Keep FCLK_CLK0 and FCLK_RESET0_N



Add FFT and DMA

- FFT for 8-point FT with floating inputs/outputs with natural order and reset.
- Add DMA





DMA

- DMA acts as interface between processor and FFT IP in FPGA
- Since FFT IP in FPGA has AXI Stream Interface, it can not write/read the data from memory directly
- DMA is configured by processor using AXI GP interface
- DMA read/writes the data from DDR using AXI Memory Mapped interface.
- DDR communicates this data with FFT using AXI stream interface.
- DMA converts AXI Memory Mapped to AXI Stream and vice-versa.

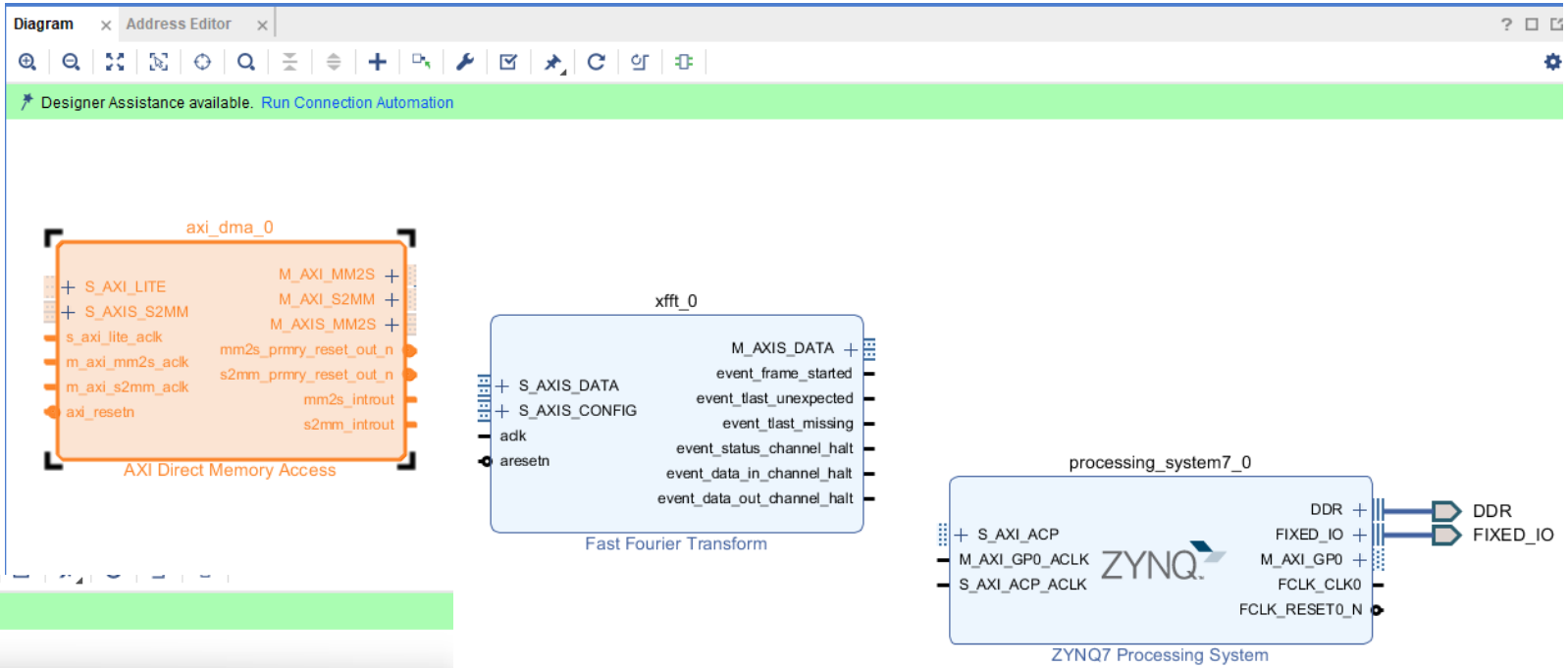
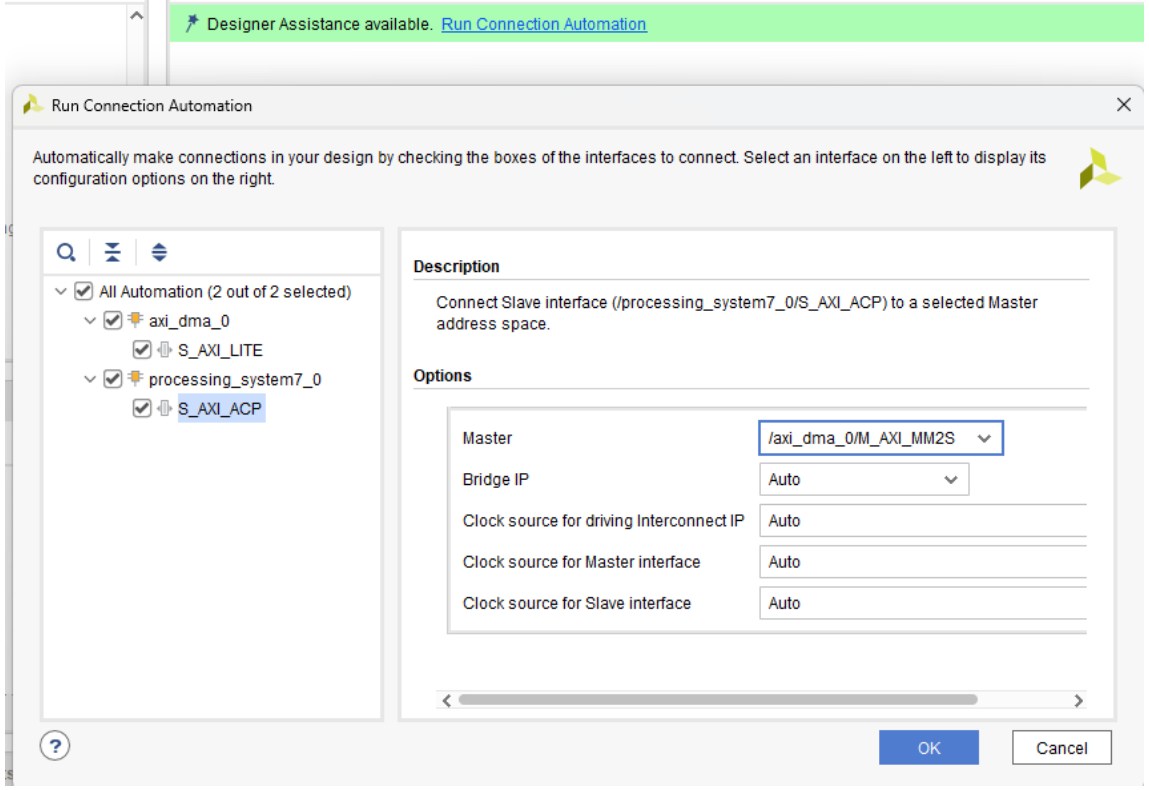


Diagram
Address Editor
Designer Assistance available. Run Connection Automation
axi_dma_0
S_AXI_LITE
S_AXIS_S2MM
s_axi_lite_aclk
m_axi_mm2s_aclk
m_axi_s2mm_aclk
axi_resetn
M_AXI_MM2S
M_AXI_S2MM
M_AXIS_MM2S
mm2s_prmry_reset_out_n
s2mm_prmry_reset_out_n
mm2s_introut
s2mm_introut
AXI Direct Memory Access
xfft_0
S_AXIS_DATA
S_AXIS_CONFIG
aclk
aresetn
M_AXIS_DATA
event_frame_started
event_tlast_unexpected
event_tlast_missing
event_status_channel_halt
event_data_in_channel_halt
event_data_out_channel_halt
Fast Fourier Transform
processing_system7_0
S_AXI_ACP
M_AXI_GP0_ACLK
S_AXI_ACP_ACLK
ZYNQ
DDR
FIXED_IO
M_AXI_GP0
FCLK_CLK0
FCLK_RESET0_N
ZYNQ7 Processing System
DDR
FIXED_IO


Designer Assistance available. Run Connection Automation
Run Connection Automation
Automatically make connections in your design by checking the boxes of the interfaces to connect. Select an interface on the left to display its configuration options on the right.
All Automation (2 out of 2 selected)
axi_dma_0
S_AXI_LITE
processing_system7_0
S_AXI_ACP
Description
Connect Slave interface (/processing_system7_0/S_AXI_ACP) to a selected Master address space.
Options
Master
/axi_dma_0/M_AXI_MM2S
Bridge IP
Auto
Clock source for driving Interconnect IP
Auto
Clock source for Master interface
Auto
Clock source for Slave interface
Auto
OK
Cancel

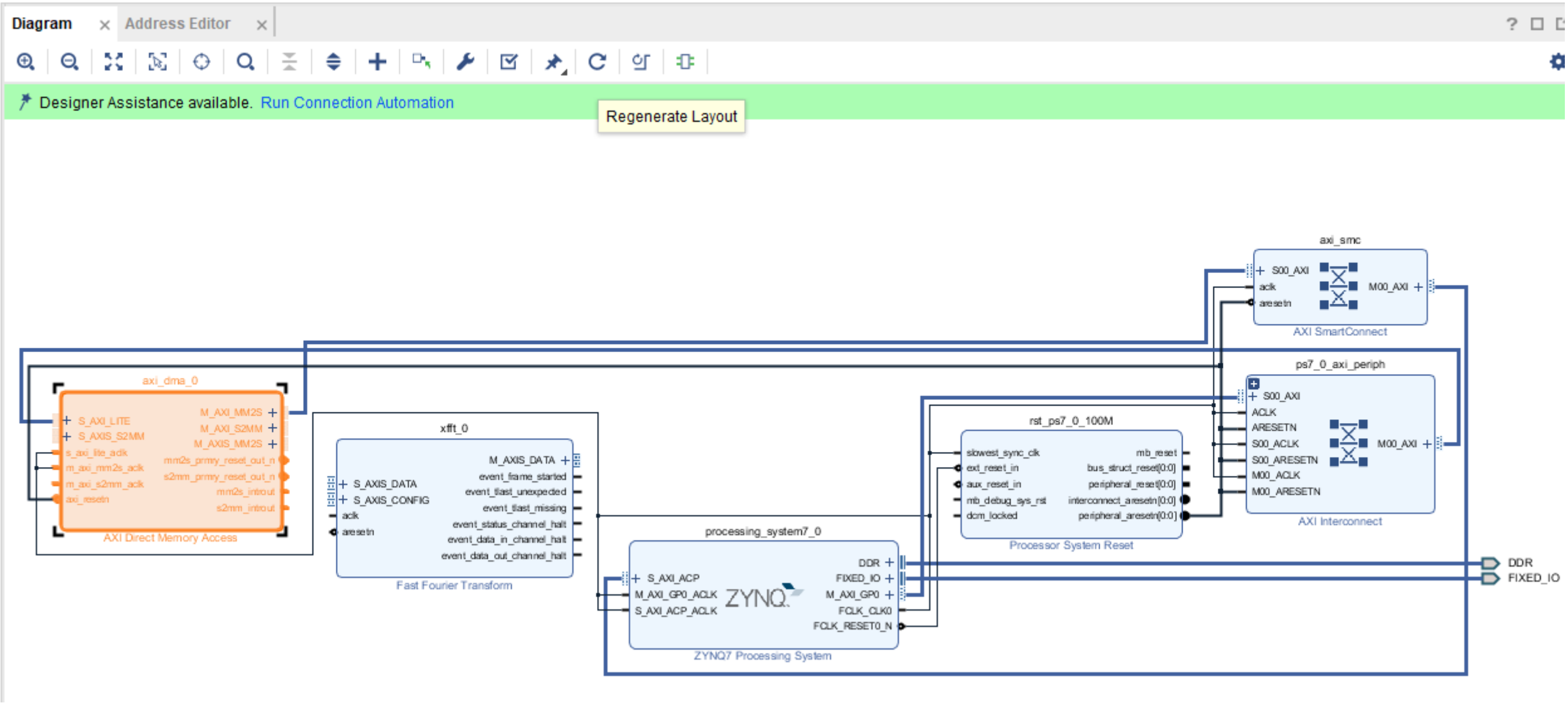


Diagram
Address Editor
Designer Assistance available. Run Connection Automation
Regenerate Layout
axi_dma_0
S_AXI_LITE
S_AXIS_S2MM
s_axi_lite_aclk
m_axi_mm2s_aclk
m_axi_s2mm_aclk
axi_resetn
M_AXI_MM2S
M_AXI_S2MM
M_AXIS_MM2S
mm2s_prmry_reset_out_n
s2mm_prmry_reset_out_n
mm2s_introut
s2mm_introut
AXI Direct Memory Access
xfft_0
S_AXIS_DATA
S_AXIS_CONFIG
aclk
aresetn
M_AXIS_DATA
event_frame_started
event_tlast_unexpected
event_tlast_missing
event_status_channel_halt
event_data_in_channel_halt
event_data_out_channel_halt
Fast Fourier Transform
processing_system7_0
S_AXI_ACP
M_AXI_GP0_ACLK
S_AXI_ACP_ACLK
ZYNQ
DDR
FIXED_IO
M_AXI_GP0
FCLK_CLK0
FCLK_RESET0_N
ZYNQ7 Processing System
rst_ps7_0_100M
slowest_sync_clk
ext_reset_in
aux_reset_in
mb_debug_sys_rst
dcm_locked
mb_reset
bus_struct_reset[0:0]
peripheral_reset[0:0]
interconnect_aresetn[0:0]
peripheral_aresetn[0:0]
Processor System Reset
axi_smc
S00_AXI
aclk
aresetn
M00_AXI
AXI SmartConnect
ps7_0_axi_periph
S00_AXI
ACLK
ARESETN
S00_ACLK
S00_ARESETN
M00_ACLK
M00_ARESETN
M00_AXI
AXI Interconnect
DDR
FIXED_IO

Regenerate Layout

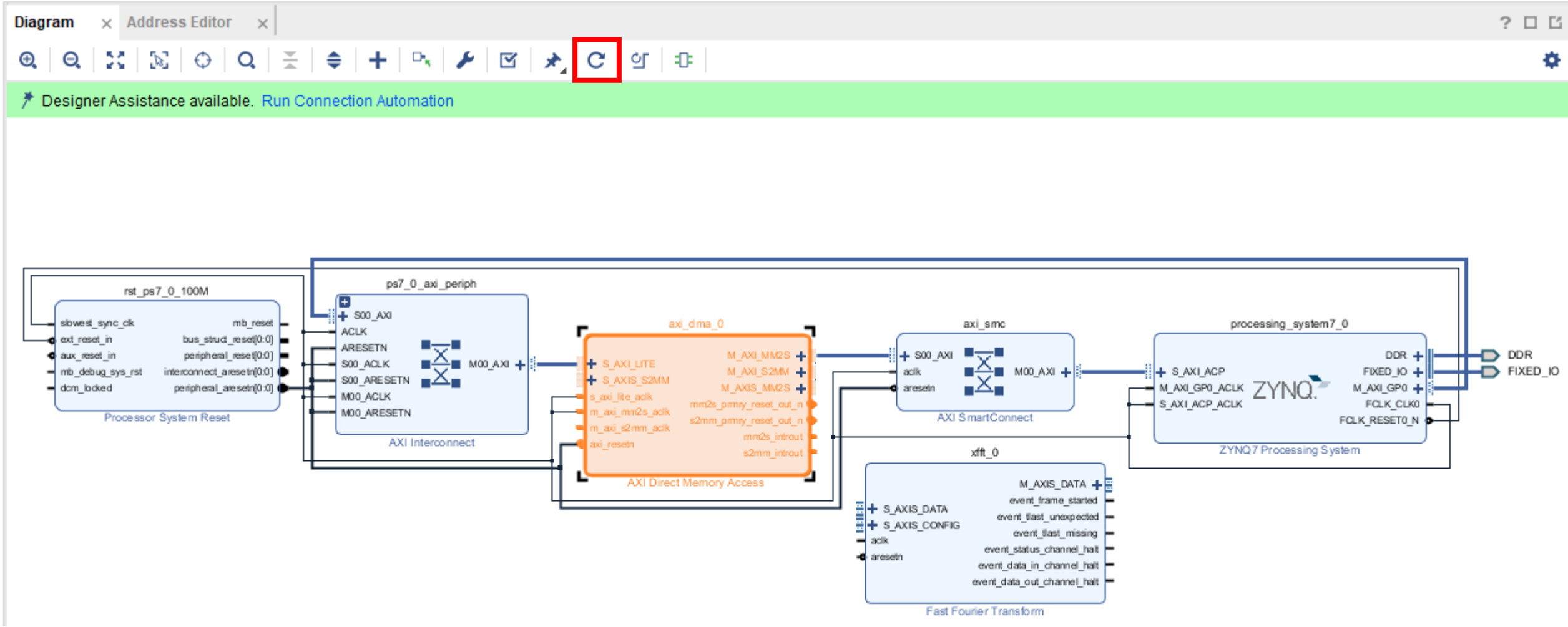


Diagram | Address Editor

Designer Assistance available. Run Connection Automation

Run Connection Automation

Automatically make connections in your design by checking the boxes of the interfaces to connect. Select an interface on the left to display its configuration options on the right.

- All Automation (1 out of 1 selected)
  - axi_dma_0
    - M_AXI_S2MM

Description

Connect Master interface (/axi_dma_0/M_AXI_S2MM) to a selected Slave interface. To connect a Master interface to an unconnected Slave interface, please use connection automation starting from the Slave Interface.

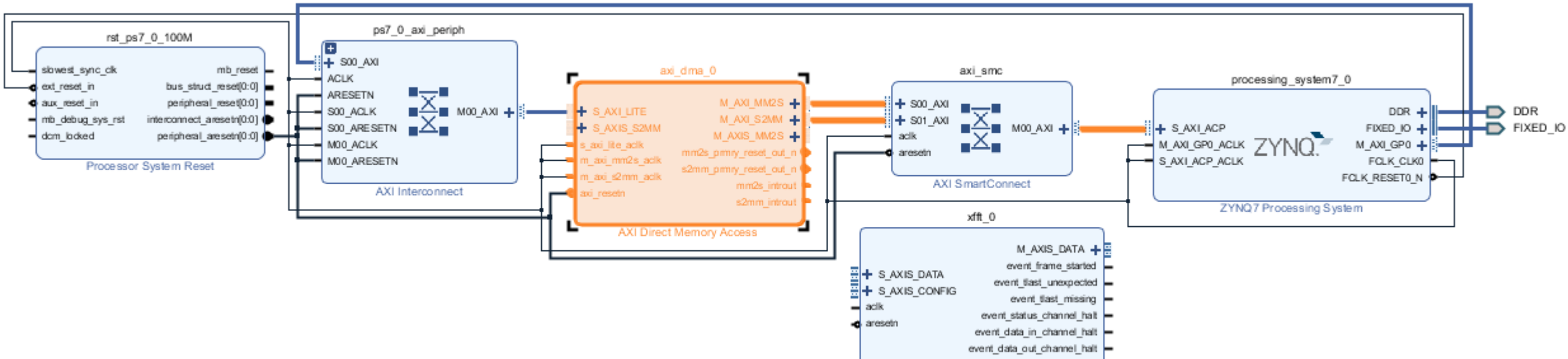
Options

Slave	/processing_system7_0/S_AXI_ACP
Bridge IP	/axi_smc
Clock source for driving Interconnect IP	/processing_system7_0/FCLK_CLK0 (100 M
Clock source for Master interface	Auto
Clock source for Slave interface	/processing_system7_0/FCLK_CLK0 (100 M

OK | Cancel

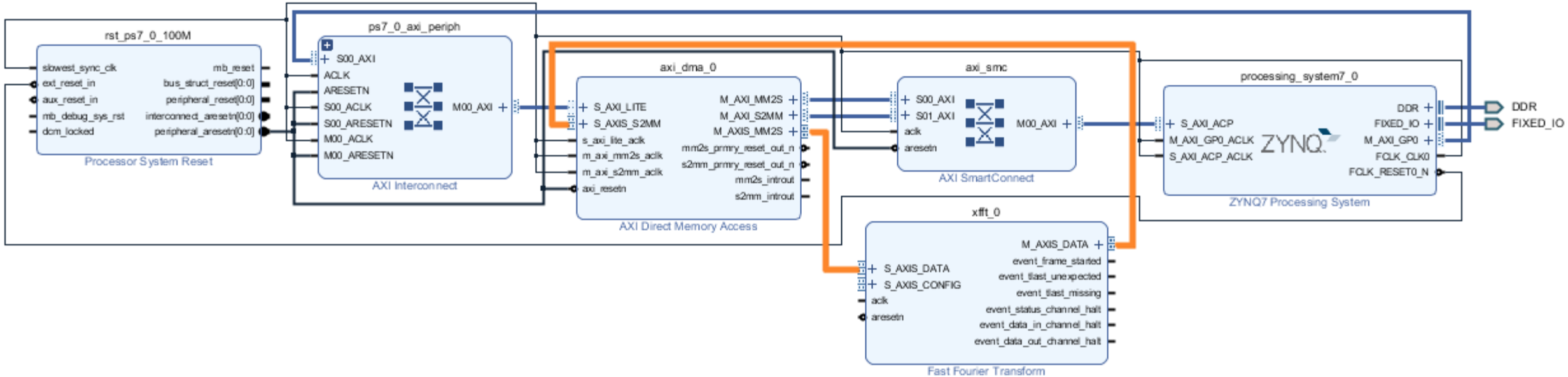
DMA <-> DDR

- DMA is connected to DDR via AXI ACP port.
- M_AXI_MM2S reads the data from memory, and forwards it to FFT over M_AXIS_MM2S stream interface
- M_AXI_MM2S writes the data to memory, obtained from FFT output over S_AXIS_MM2S stream interface



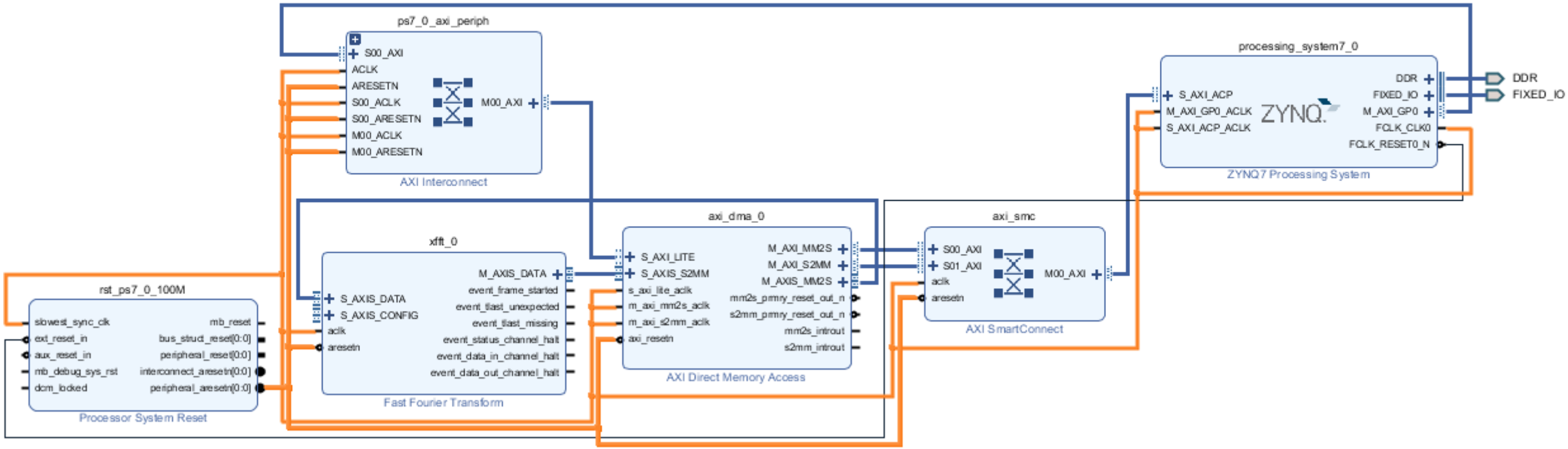
AXI DMA <-> FFT

- AXI DMA M_AXIS_MM2S is connected to S_AXIS_Data of FFT. This means stream output of the data read by DMA is passed to stream input of FFT for processing.
- AXI DMA S_AXIS_S2MM is connected to M_AXIS_Data of FFT. This means data processed by FFT is passed to DMA to write back to memory.



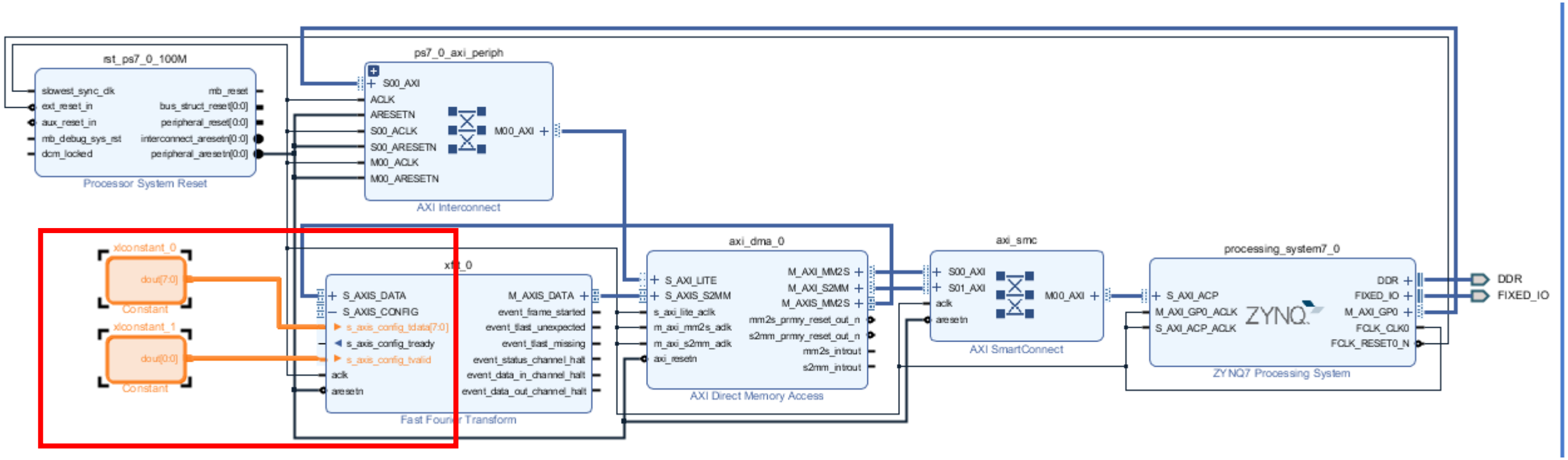
FFT Clock and Reset Signals

- Connect clock and reset inputs of FFT to rest of the clock and reset signals.



FFT Configuration

- Similar to testbench, we are directly connecting config_data and config_valid to 1 using Constant IP.
- Since config is AXI Stream interface, you can not use GP port

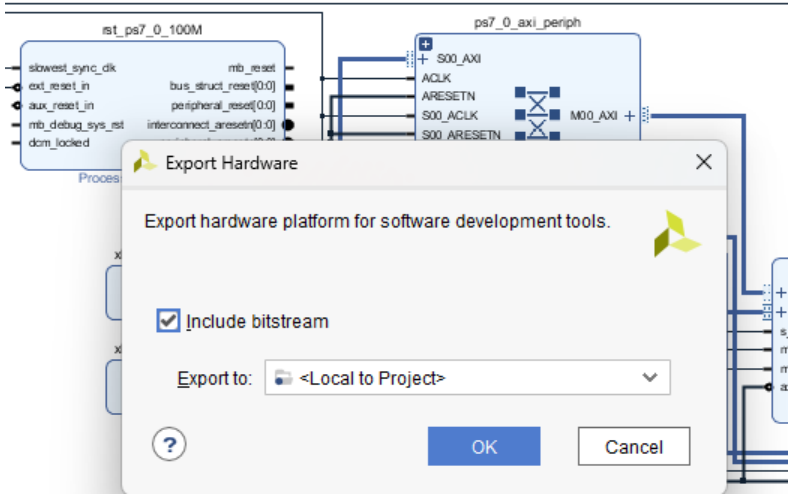


Block Diagram

- Validate the design
- Create HLD Wrapper
- Generate output products
- Generate Bitstream
- Export with Bitstream
- Launch SDK

Diagram | Address Editor

Cell	Slave Interface	Base Name	Offset Address	Range	High Address
processing_system7_0					
Data (32 address bits : 0x40000000 [1G])					
axi_dma_0	S_AXI_LITE	Reg	0x4040_0000	64K	0x4040_FFFF
axi_dma_0					
Data_MM2S (32 address bits : 4G)					
processing_system7_0	S_AXI_ACP	ACP_DDR_LOWOCM	0x0000_0000	512M	0x1FFF_FFFF
Excluded Address Segments (2)					
processing_system7_0	S_AXI_ACP	ACP_IOP	0xE000_0000	4M	0xE03F_FFFF
processing_system7_0	S_AXI_ACP	ACP_M_AXI_GP0	0x4000_0000	1G	0x7FFF_FFFF
Data_S2MM (32 address bits : 4G)					
processing_system7_0	S_AXI_ACP	ACP_DDR_LOWOCM	0x0000_0000	512M	0x1FFF_FFFF
Excluded Address Segments (2)					
processing_system7_0	S_AXI_ACP	ACP_IOP	0xE000_0000	4M	0xE03F_FFFF
processing_system7_0	S_AXI_ACP	ACP_M_AXI_GP0	0x4000_0000	1G	0x7FFF_FFFF



SDK

- Create HelloWorld Project
- Modify to implement 8-point FFT on PS
- Modify to include DMA initialization (Use dma_init.h) and configuration
- Compare FFT output of PS and PL
- Compare execution time

PS FFT

```c
#include <stdio.h>
#include <complex.h>
#include <stdlib.h>
#include "platform.h"
#include "xil_printf.h"
#include <xtime_l.h>
#include "xparameters.h"
#include "xaxidma.h"
#include "dma_init.h"

#define N 8

const int rev8[N] = {0,4,2,6,1,5,3,7};
const float complex W[N/2] = {1-0*I,0.7071067811865476-0.7071067811865475*I,0.0-1*I,-0.7071067811865476-0.7071067811865475*I};

void bitreverse(float complex dataIn[N], float complex dataOut[N]){
    bit_reversal: for(int i=0;i<N;i++){
        dataOut[i]=dataIn[rev8[i]];
    }
}

void FFT_stages(float complex FFT_input[N],float complex FFT_output[N]){
    float complex temp1[N], temp2[N];
    stage1: for(int i=0;i<N;i=i+2){
            temp1[i] = FFT_input[i]+FFT_input[i+1];
            temp1[i+1] = FFT_input[i]-FFT_input[i+1];
    }
    stage2: for(int i=0;i<N;i=i+4){
                for(int j=0;j<2;++j){
                    temp2[i+j] = temp1[i+j]+W[2*j]*temp1[i+j+2];
                    temp2[i+2+j] =temp1[i+j]-W[2*j]*temp1[i+j+2];
            }
    }
    stage3: for(int i=0;i<N/2;i=i+1){
                FFT_output[i]=temp2[i]+W[i]*temp2[i+4];
                FFT_output[i+4]=temp2[i]-W[i]*temp2[i+4];
        }
}
```

Main: FFT in PS

```c
int main()
{
    init_platform();
    // Initializing Timer instances for PS and PL
    XTime PL_start_time,PL_end_time;
    XTime PS_start_time,PS_end_time;
    // Initializing software and hardware output buffers
    const float complex FFT_input[N] = {11+23*I,32+10*I,91+94*I,15+69*I,47+96*I,44+12*I,96+17*I,49+58*I};
    float complex FFT_output_sw[N],FFT_output_hw[N];
    float complex FFT_rev_sw[N];

    ///////////// Software 8-point FFT
    XTime_SetTime(0); // Setting Timer to value 0
    XTime_GetTime(&PS_start_time);// Get Start Time
    bitreverse(FFT_input,FFT_rev_sw);
    FFT_stages(FFT_rev_sw,FFT_output_sw);
    XTime_GetTime(&PS_end_time);// Get End Time
```

Main: FFT in PL via DMA

```
///////////// Hardware 8-point FFT
int status;
XAxiDma AxiDMA;
status=DMA_Init(&AxiDMA, XPAR_AXI_DMA_0_DEVICE_ID);
if(status)
    return 1;// DMA Init Failed

XTime_SetTime(0);// Setting Timer to value 0
XTime_GetTime(&PL_start_time);// Get Start Time
// Simple DMA Transfers
status=XAxiDma_SimpleTransfer(&AxiDMA,(UINTPTR)FFT_output_hw,(sizeof(float complex)*N),XAXIDMA_DEVICE_TO_DMA);
status=XAxiDma_SimpleTransfer(&AxiDMA,(UINTPTR)FFT_input,(sizeof(float complex)*N),XAXIDMA_DMA_TO_DEVICE);

// POLLING-Check whether the DMA-to-Device and Device-to-DMA transfers are complete
while(XAxiDma_Busy(&AxiDMA,XAXIDMA_DMA_TO_DEVICE));
/    printf("\n\rDMA-to-Device Transfer Done!");
while(XAxiDma_Busy(&AxiDMA,XAXIDMA_DEVICE_TO_DMA));
//   printf("\n\rDevice-to-DMA Transfer Done!");
XTime_GetTime(&PL_end_time);// Get End Time
```

Main: Compare PS and PL outputs

```
/////////// Verifying Hardware result with Software
for(int i=0;i<N;i++){
    printf("\n\rPS Output- %f+%fI, PL Output- %f+%fI",crealf(FFT_output_sw[i]),cimagf(FFT_output_sw[i]),crealf(FFT_output_hw[i]),cimagf(FFT_output_hw[i]));
    float diff1=abs(crealf(FFT_output_sw[i])-crealf(FFT_output_hw[i]));
    float diff2=abs(cimagf(FFT_output_sw[i])-cimagf(FFT_output_hw[i]));
    if(diff1>=0.0001 && diff2>=0.0001){
        printf("\n\rData Mismatch found at index %d !",i);
        break;
    }
    else
        printf("DMA Transfer Successful!");
}

/////////// Software & Hardware Exceution Time calculation
printf("\n\r------- Execution Time Comparison --------");
float time=0;
time= (float)1.0 * (PS_end_time-PS_start_time)/(COUNTS_PER_SECOND/1000000);
printf("\n\rExecution time for PS in Micro-seconds: %f",time);
// Hardware Exceution Time calculation
time=0;
time= (float)1.0 * (PL_end_time-PL_start_time)/(COUNTS_PER_SECOND/1000000);
printf("\n\rExecution time for PL in Micro-seconds: %f",time);

return 0;
```

Execution Time Using Timer

```
Terminal requirements :
  (i) Processor's STDOUT is redirected to the ARM DCC/MDM UART
  (ii) Processor's STDIN is redirected to the ARM DCC/MDM UART.
       Then, text input from this console will be sent to DCC/MDM's UART port.
  NOTE: This is a line-buffered console and you have to press "Enter"
        to send a string of characters to DCC/MDM.



PS Output- 385.000000+379.000000I, PL Output- 385.000000+379.000000IDMA Transfer Successful!

PS Output- 62.920311+-44.665474I, PL Output- 62.920311+-44.665474IDMA Transfer Successful!

PS Output- -234.000000+-4.000000I, PL Output- -234.000000+-4.000000IDMA Transfer Successful!

PS Output- -122.192383+-36.280701I, PL Output- -122.192390+-36.280701IDMA Transfer Successful!

PS Output- 105.000000+81.000000I, PL Output- 105.000000+81.000000IDMA Transfer Successful!

PS Output- 19.079691+-91.334526I, PL Output- 19.079689+-91.334526IDMA Transfer Successful!

PS Output- -24.000000+20.000000I, PL Output- -24.000000+20.000000IDMA Transfer Successful!

PS Output- -103.807617+-119.719299I, PL Output- -103.807610+-119.719299IDMA Transfer Successful!

------- Execution Time Comparison --------

Execution time for PS in Micro-seconds: 4.707692

Execution time for PL in Micro-seconds: 4.584615
```