LAB ASSIGNMENT -4

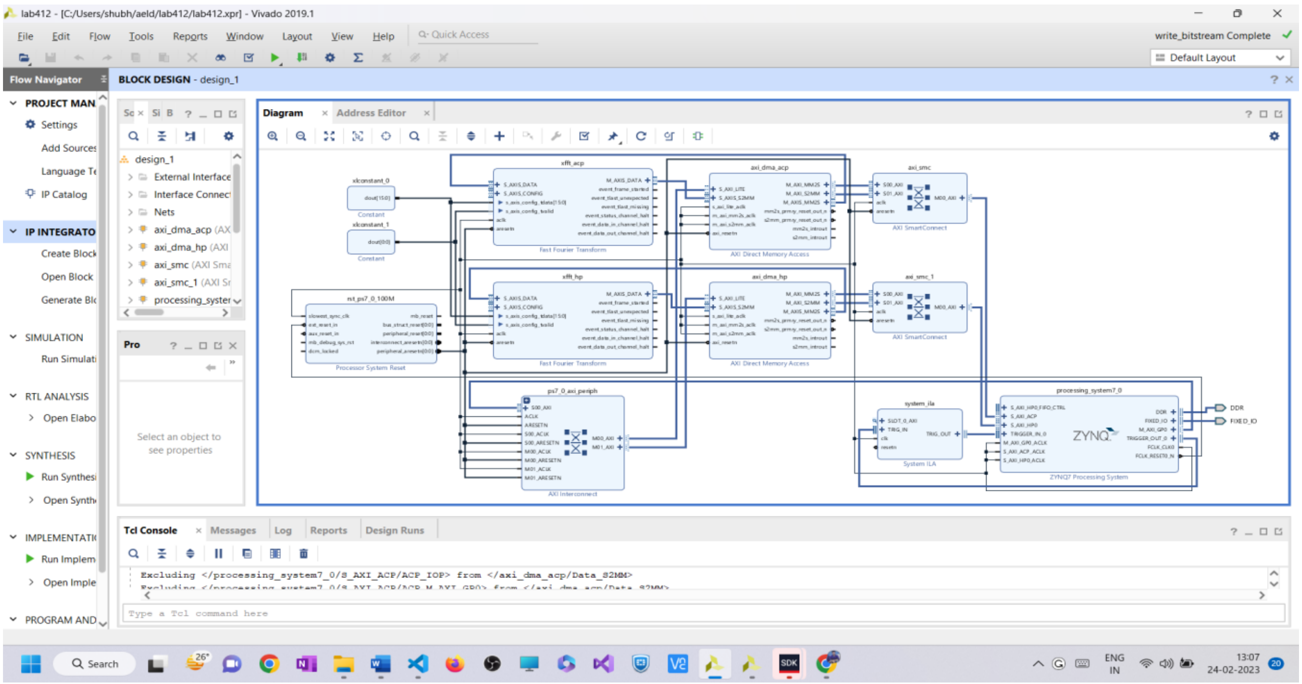
AELD

NAME – ARYAN GUPTA

ROLL NO – MT22154

For 512 FFT

Block Diagram



Code

```
#include <stdio.h>
#include <stdlib.h>
#include <complex.h>
#include <xtime_l.h>
#include "xparameters.h"
#include "xaxidma.h"
#include "src_fftsw.h"
#define FFT_Size 512
#define reverse(n) ((n & 0x1) << 8) | ((n & 0x2) << 6) | ((n & 0x4) << 4) | ((n &
0x8) << 2) | ((n & 0x10) << 0) | ((n & 0x20) >> 0) | ((n & 0x40) >> 2) | ((n &
0x80) >> 4) | ((n & 0x100) >> 6) | ((n & 0x200) >> 8)
```

```
void FFT_sw(float FFTIn_I[FFT_Size], float FFTIn_R[FFT_Size], float
FFTOut_I[FFT_Size], float FFTOut_R[FFT_Size])
       {

         DTYPE temp_R;   /*temporary storage complex variable*/
         DTYPE temp_I;   /*temporary storage complex variable*/

         int i,j;      /* loop indexes */
         int i_lower;    /* Index of lower point in butterfly */

         int stage;
         int subFFTSize; //Size of FFT in each stage of FFT
         int BFWidth;      /*Butterfly Width*/

         /*=====================BEGIN BIT REBERSAL===========================*/
         for (i = 0; i < FFT_Size; ++i) {
           FFTOut_R[reverse(i)] = FFTIn_R[i];
           FFTOut_I[reverse(i)] = FFTIn_I[i];
         }
         /*+++++++++++++++++++++END OF BIT REVERSAL+++++++++++++++++++++++++*/

         /*=======================BEGIN: FFT========================*/
         // Do FFTSTAGES of butterflies
         DTYPE BFWeight_R, BFWeight_I;
        // For N-point FFT, there are log2(N) stages
         stages:for(stage=1; stage<= FFTSTAGES; stage++)
         {
           subFFTSize = 1 << stage;    // DFT = 2^stage = points in sub DFT
           BFWidth = subFFTSize >> 1;       // Butterfly WIDTHS in sub-FFT
(FFTSIZE of sub-FFT/2) no of weights

           // Perform butterflies for j-th stage
           // This loop runs for the iteration equal to BF width
           // In 4-point FFT, BF width is 1 in stage 1 and 2 in stage 2
           // In 8-point FFT, BF width is 1 in stage 1, 2 in stage 2 and 4 in
stage 3
           butterfly:for(j=0; j<BFWidth; j++)
           {
             //Note that weights of all butterfly units are same in a given stage
             // We can reduce the number of memory read by using this for loop
             BFWeight_R = W_real[j * (FFT_Size>>stage)];
             BFWeight_I = W_imag[j * (FFT_Size>>stage)];

           // This loop is for all butterflies in a stage that use same W**k
           // In 4-point FFT, we have two BFs in stage 1
           // In 8-point FFT, we have four BFs in stage 1 and two BFs in stage 2
             // Each butter fly weight affects two outputs and hence we have two
outputs for each iteration
           subDFTSize:for(i =j ; i < FFT_Size; i += subFFTSize) // This loop runs
for FFTSIZE/SubFFTSize iterations
             {
             i_lower = i + BFWidth;      //index of lower point in butterfly
             temp_R = FFTOut_R[i_lower] * BFWeight_R - FFTOut_I[i_lower] *
BFWeight_I;
             temp_I = FFTOut_I[i_lower] * BFWeight_R + FFTOut_R[i_lower] *
BFWeight_I;

             FFTOut_R[i_lower] = FFTOut_R[i] - temp_R;
             FFTOut_I[i_lower] = FFTOut_I[i] - temp_I;
```

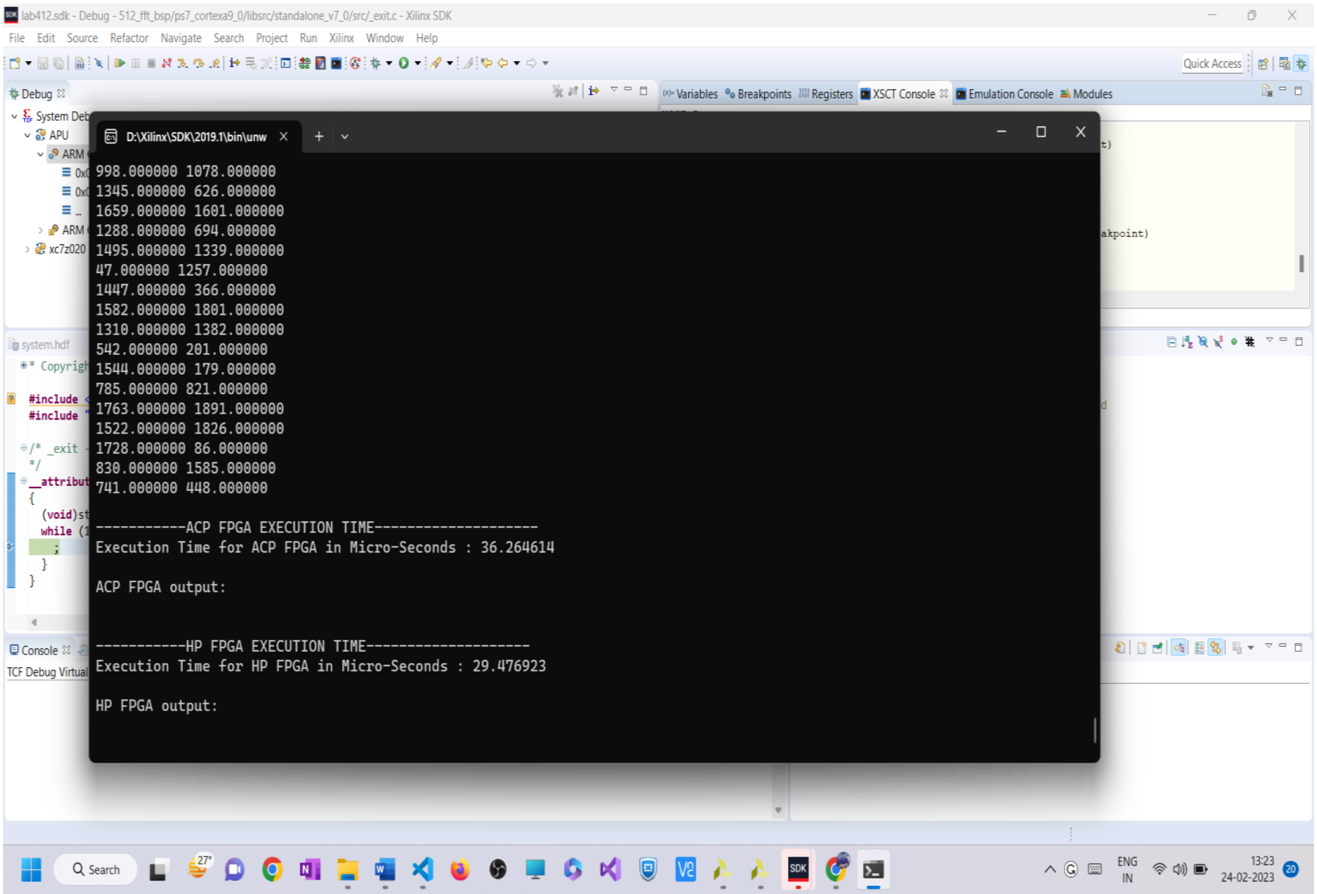
```
            FFTOut_R[i] = FFTOut_R[i] + temp_R;
            FFTOut_I[i] = FFTOut_I[i] + temp_I;
            }
        }
        }
      }

int FFTPS()
{
             float DataIN_R[FFT_Size];
           float DataIN_I[FFT_Size];
           float complex FFTIn_C[FFT_Size];
           for (int k = 0; k<FFT_Size;k++)
           {
             FFTIn_C[k] = (k/4) + (k/2)*I;
             DataIN_R[k] = creal(FFTIn_C[k]);
             DataIN_I[k] = cimag(FFTIn_C[k]);
           }
           float FFTOut_R[FFT_Size];
           float FFTOut_I[FFT_Size];
           XTime time_PS_start , time_PS_end;
           XTime_SetTime(0);
           XTime_GetTime(&time_PS_start);
           FFT_sw(DataIN_I, DataIN_R, FFTOut_I,FFTOut_R);
           XTime_GetTime(&time_PS_end);
           printf("\n FFT output: \r\n");
//         for (int j = 0 ; j < FFT_Size ; j++)
//         {
//             printf("PS Output : %f + I%f \n " , FFTOut_R[j], FFTOut_I[j]);
//             usleep(0.1); // Always add some buffer time between display
//         }
           printf("\n-----------EXECUTION TIME--------------------\n");
           float time_processor = 0;
           time_processor = (float)1.0 * (time_PS_end - time_PS_start) /
(COUNTS_PER_SECOND/1000000);
           printf("Execution Time for PS in Micro-Seconds : %f\n" ,
time_processor);
           //     printf("======================FFT Input==================\n");
           //     for (int i = 0 ; i < FFT_Size ; i++)
           //             {
           //                   printf("%f %f\n", crealf(FFT_input[i]),
cimagf(FFT_input[i]));
           //             }
           return 0;
}
int FFTHPVSACP()
{
       float complex FFT_input[FFT_Size];
               float temp_r, temp_i;
                XTime seed_value;
                XTime_GetTime(&seed_value);
                srand(seed_value);
                for (int i = 0 ; i < FFT_Size ; i++)
                 {
                     temp_r = rand()%2000;
                     temp_i = rand()%2000;
                     FFT_input[i] = temp_r + I*temp_i;
                 }
    float complex FFT_ACPoutput[FFT_Size];
```

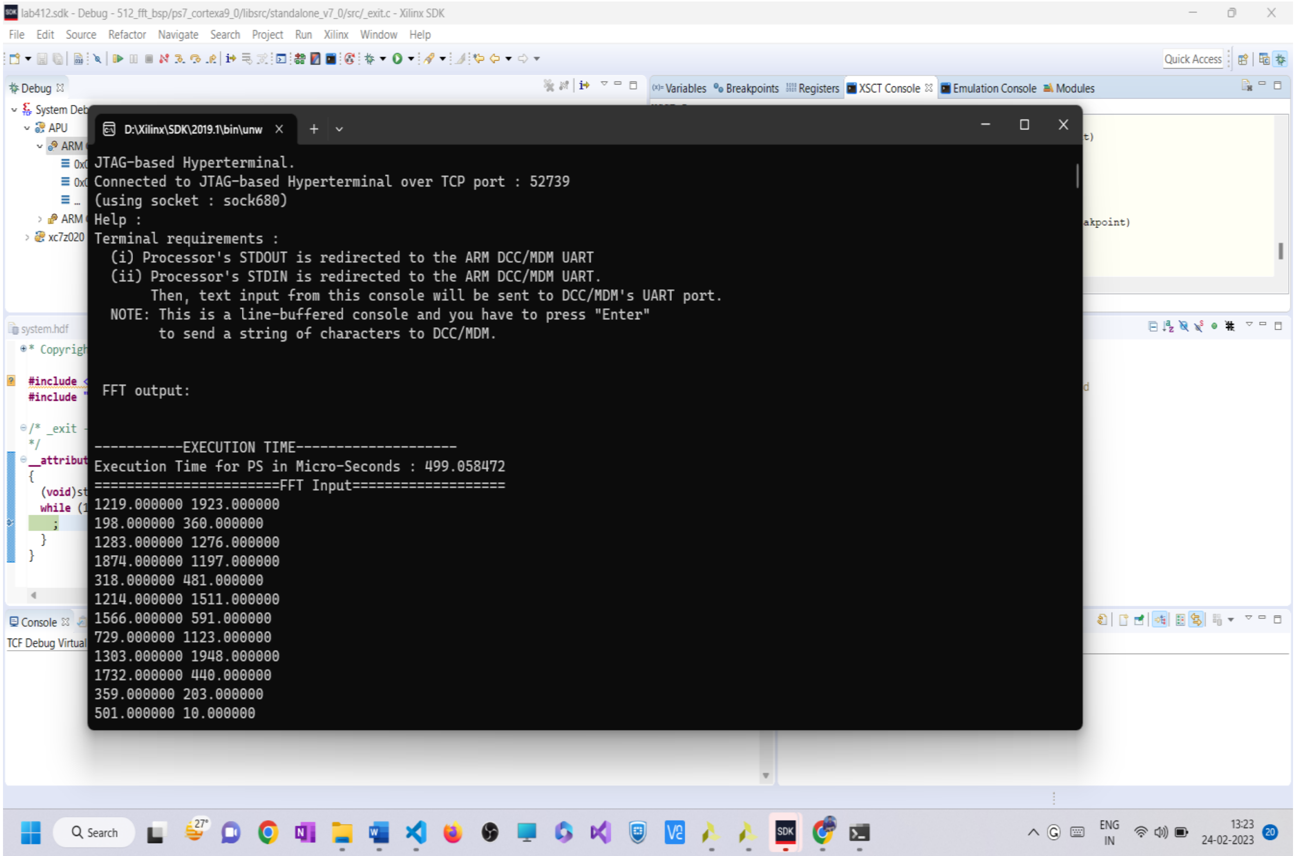
```c
    XAxiDma_Config *DMAACP_confptr;
    DMAACP_confptr = XAxiDma_LookupConfig(XPAR_AXI_DMA_ACP_DEVICE_ID);
    int status;
    XAxiDma DMAACP_instance;
    status = XAxiDma_CfgInitialize(&DMAACP_instance, DMAACP_confptr);
    if(status!=XST_SUCCESS)
    {
        printf("ACP DMA Init Failed\n");
        return 1;
    }
    XTime time_ACP_start , time_ACP_end;
    XTime_SetTime(0);
    XTime_GetTime(&time_ACP_start);
//    Xil_DCacheFlushRange((UINTPTR)FFT_input,(sizeof(float)*FFT_Size));
//    Xil_DCacheFlushRange((UINTPTR)FFT_input1,(sizeof(float)*FFT_Size));
    status = XAxiDma_SimpleTransfer(&DMAACP_instance, (UINTPTR)FFT_ACPoutput,
FFT_Size*2*sizeof(float), XAXIDMA_DEVICE_TO_DMA);
    if(status!=XST_SUCCESS)
    {
        printf("ACP DMA Device to DMA Configuration Failed\n");
        return 1;
    }
    status = XAxiDma_SimpleTransfer(&DMAACP_instance, (UINTPTR)FFT_input,
FFT_Size*2*sizeof(float), XAXIDMA_DMA_TO_DEVICE);
    if(status!=XST_SUCCESS)
    {
        printf("ACP DMA DMA to Device Configuration Failed\n");
        return 1;
    }
//    Xil_DCacheInvalidateRange((UINTPTR)FFT_input,(sizeof(float)*FFT_Size));
    status = XAxiDma_ReadReg(XPAR_AXI_DMA_ACP_BASEADDR,0x04);
    status = status & 0x00000002;
    while (status!= 0x00000002)
    {
        status = XAxiDma_ReadReg(XPAR_AXI_DMA_ACP_BASEADDR,0x04);
        status = status & 0x00000002;
    }
    status = XAxiDma_ReadReg(XPAR_AXI_DMA_ACP_BASEADDR,0x34);
    status = status & 0x00000002;
    while (status!= 0x00000002)
    {
        status = XAxiDma_ReadReg(XPAR_AXI_DMA_ACP_BASEADDR,0x34);
        status = status & 0x00000002;
    }
    XTime_GetTime(&time_ACP_end);
    printf("\n-----------ACP FPGA EXECUTION TIME--------------------\n");
        float time_ACPFPGA = 0;
        time_ACPFPGA = (float)1.0 * (time_ACP_end - time_ACP_start) /
(COUNTS_PER_SECOND/1000000);
        printf("Execution Time for ACP FPGA in Micro-Seconds : %f\n" ,
time_ACPFPGA);
        printf("\nACP FPGA output: \r\n");
//        for (int i = 0 ; i < FFT_Size ; i++)
//        {
//            printf("%f %f\n", crealf(FFT_ACPoutput[i]),
cimagf(FFT_ACPoutput[i]));
//        }
                  XTime time_HP_start, time_HP_end;
                 float complex FFT_HPoutput[FFT_Size];
```

```
                     XAxiDma_Config *DMAHP_confptr;
                     DMAHP_confptr =
XAxiDma_LookupConfig(XPAR_AXI_DMA_HP_DEVICE_ID);
                     int status1;
                     XAxiDma DMAHP_instance;
                     status1 = XAxiDma_CfgInitialize(&DMAHP_instance,
DMAHP_confptr);
                     if(status1!=XST_SUCCESS)
                     {
                     printf("HP DMA Init Failed\n");
                     return 1;
                     }
                     XTime_SetTime(0);
                         XTime_GetTime(&time_HP_start);

Xil_DCacheFlushRange((UINTPTR)FFT_input,(sizeof(float)*FFT_Size));
                         status1 = XAxiDma_SimpleTransfer(&DMAHP_instance,
(UINTPTR)FFT_HPoutput, FFT_Size*2*sizeof(float), XAXIDMA_DEVICE_TO_DMA);
                         if(status1!=XST_SUCCESS)
                         {
                             printf("HP DMA Device to DMA Configuration
Failed\n");
                             return 1;
                         }
                         status1 = XAxiDma_SimpleTransfer(&DMAHP_instance,
(UINTPTR)FFT_input, FFT_Size*2*sizeof(float), XAXIDMA_DMA_TO_DEVICE);
                         if(status1!=XST_SUCCESS)
                         {
                             printf("HP DMA DMA to Device Configuration
Failed\n");
                             return 1;
                         }

Xil_DCacheInvalidateRange((UINTPTR)FFT_input,(sizeof(float)*FFT_Size));
                         status1 =
XAxiDma_ReadReg(XPAR_AXI_DMA_HP_BASEADDR,0x04);
                         status1 = status1 & 0x00000002;
                         while (status1!= 0x00000002)
                         {
                             status1 =
XAxiDma_ReadReg(XPAR_AXI_DMA_HP_BASEADDR,0x04);
                             status1 = status1 & 0x00000002;
                         }
                         status1 =
XAxiDma_ReadReg(XPAR_AXI_DMA_HP_BASEADDR,0x34);
                         status1 = status1 & 0x00000002;
                         while (status1!= 0x00000002)
                         {
                             status1 =
XAxiDma_ReadReg(XPAR_AXI_DMA_HP_BASEADDR,0x34);
                             status1 = status1 & 0x00000002;
                         }
                         XTime_GetTime(&time_HP_end);
                         printf("\n-----------HP FPGA EXECUTION TIME-------------
-------\n");
                             float time_HPFPGA = 0;
                             time_HPFPGA = (float)1.0 * (time_HP_end -
time_HP_start) / (COUNTS_PER_SECOND/1000000);
```

```
        printf("Execution Time for HP FPGA in Micro-Seconds : %f\n" , time_HPFPGA);

        printf("\nHP FPGA output: \r\n");
//        for (int i = 0 ; i < FFT_Size ; i++)
//        {
//            printf("%f %f\n", crealf(FFT_HPoutput[i]), cimagf(FFT_HPoutput[i]));
//        }

}
int main()
{
    init_platform();
    FFTPS();
    FFTHPVSACP();
    cleanup_platform();
}
```

Jtagterminal




lab412.sdk - Debug - 512_fft_bsp/ps7_cortexa9_0/libsrc/standalone_v7_0/src/_exit.c - Xilinx SDK
File Edit Source Refactor Navigate Search Project Run Xilinx Window Help
D:\Xilinx\SDK\2019.1\bin\unw
JTAG-based Hyperterminal.
Connected to JTAG-based Hyperterminal over TCP port : 52739
(using socket : sock680)
Help :
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.
FFT output:
-----------EXECUTION TIME--------------------
Execution Time for PS in Micro-Seconds : 499.058472
=====================FFT Input==================
1219.000000 1923.000000
198.000000 360.000000
1283.000000 1276.000000
1874.000000 1197.000000
318.000000 481.000000
1214.000000 1511.000000
1566.000000 591.000000
729.000000 1123.000000
1303.000000 1948.000000
1732.000000 440.000000
359.000000 203.000000
501.000000 10.000000
13:23
24-02-2023