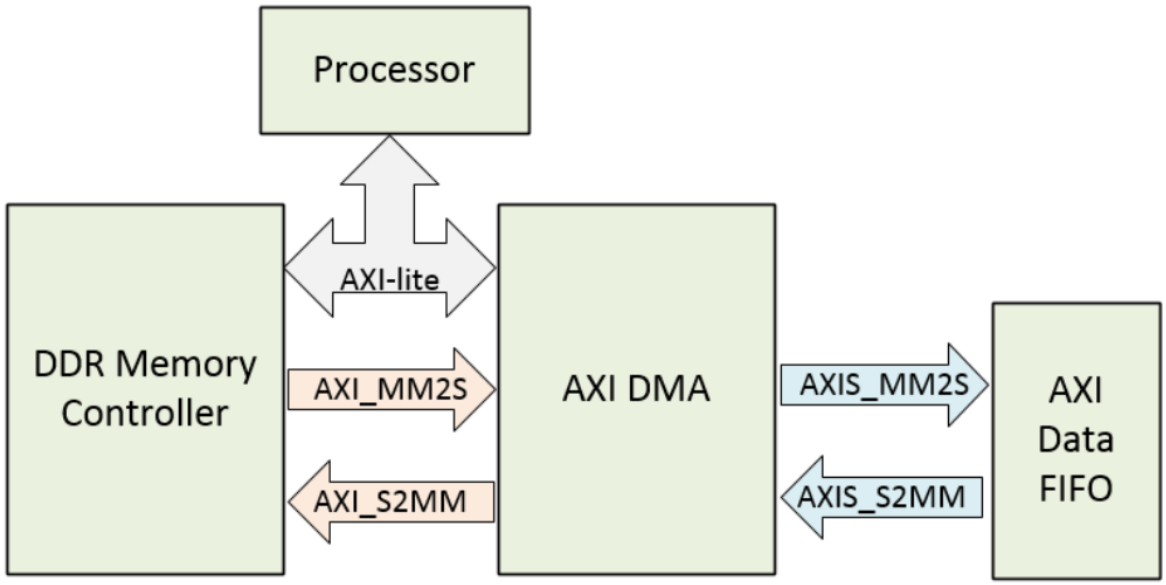
LAB ASSIGNMENT -3

AELD

NAME – ARYAN GUPTA

ROLL NO – MT22154

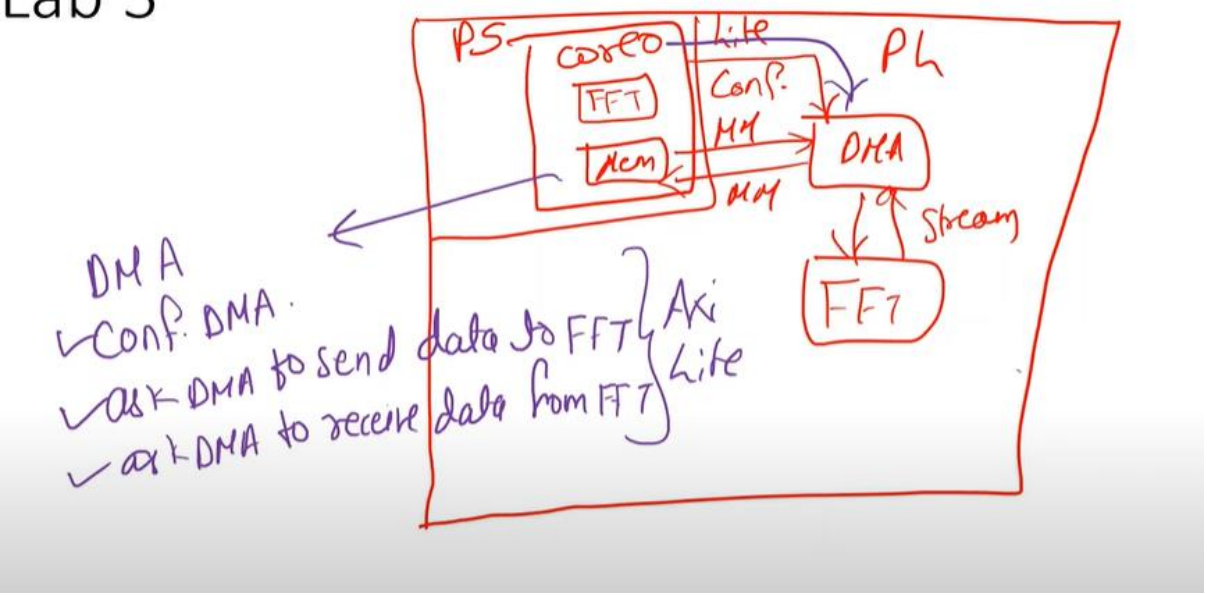
In this lab, I do the FFT through the hardware. But as we know, we cannot directly connect the memory to the hardware (That is FFT in this case), .so an intermediator comes into the picture, which is the DMA (for Direct Memory Access).



Source:- https://www.fpgadeveloper.com/2014/08/using-the-axi-dma-in-vivado.html/

AXI Data FIFO is not used in the implementation.

Lab 3



Source: - Sumit Sir Lab Handout

AXI DMA – FFT connected through stream.

AXI DMA – Memory – Memory Map Interface

AXI DMA config – through the AXI Lite Interface

Hardware changes -> 8 FFT -> FFT hardware size 8.

For the 512 FFT -> FFT hardware size 512.

I use here the burst transfer of 256.

And the unaligned transfer is allowed.

Stream transfer is 64.

Hardware is always faster than software, so let us verify that.

I must also generate the bitstream file here because we will program FPGA.

So, firstly we need to configure the AXI DMA whether i can do it by using the Device ID or the Base Address, defined in the xparameters.h file.

And I also need to look in the xaxidma.h file for the functions and definition of dma transfer and other queries.

That you can find this file in the BSP folder -> Includes.

Firstly, I must define a pointer -> of XAxiDma_Config type (which holds the config).

Now I pass it the function XAxiDma_LookupConfig(can give device id or base address).

Now I define the status of config -> whether a success or failure.

XAxiDma_CfgInitialize(instance address (as we define it with not *,XAxiDma_Config type)

So, if there is a success, then the status will get the XST_SUCCESS, else not.

Now after that, I do the simple transfer by using the simple transfer function.

What a simple transfer function does is that it calls with these parameters:- (XAxiDma *InstancePtr, UINTPTR BuffAddr, u32 Length, int Direction)

InstancePtr -> for instance, pointer

BuffAddr -> Where to write.

U32 Length -> Size of transfer

Direction -> from where to where.

So, what I do here is the data transfer from the Device to DMA and then from DMA to the device.

So that I can tell the hardware that we will transfer that much amount of data.

In addition, I also check the status of the things we do in between.

After that, I call the XAxiDma_ReadReg to check the values of the registers and also check the offsets.

Address Space Offset (1)	Name	Description
00h	MM2S_DMACR	MM2S DMA Control register
04h	MM2S_DMASR	MM2S DMA Status register
08h	MM2S_CURDESC	MM2S Current Descriptor Pointer. Lower 32 bits of the address.
0Ch	MM2S_CURDESC_MSB	MM2S Current Descriptor Pointer. Upper 32 bits of address.
10h	MM2S_TAILDESC	MM2S Tail Descriptor Pointer. Lower 32 bits.
14h	MM2S_TAILDESC_MSB	MM2S Tail Descriptor Pointer. Upper 32 bits of address.
2Ch (2)	SG_CTL	Scatter/Gather User and Cache
30h	S2MM_DMACR	S2MM DMA Control register
34h	S2MM_DMASR	S2MM DMA Status register
38h	S2MM_CURDESC	S2MM Current Descriptor Pointer. Lower 32 address bits
3Ch	S2MM_CURDESC_MSB	S2MM Current Descriptor Pointer. Upper 32 address bits.
40h	S2MM_TAILDESC	S2MM Tail Descriptor Pointer. Lower 32 address bits.
44h	S2MM_TAILDESC_MSB	S2MM Tail Descriptor Pointer. Upper 32 address bits.

Source: - Taken from Xilinx DMA dataset file

I use here.

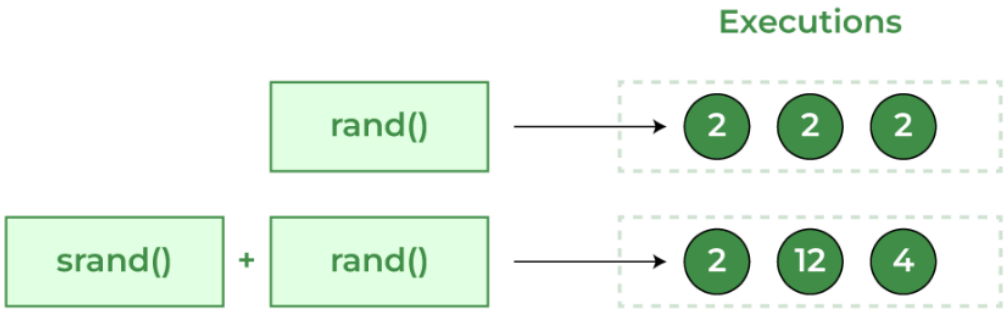
1) 0x04 – Check MM2S (Memory map to stream) DMA status register
2) 0x34 – Check S2MM (Stream to Memory Map) DMA status register

I take input using the srand and rand functions.

So, what srand does -> it sets the starting point for producing a series of pseudo-random numbers.

And rand -> it generates the random numbers.

So might well be a difference in the values of rand when srand is used and not used.

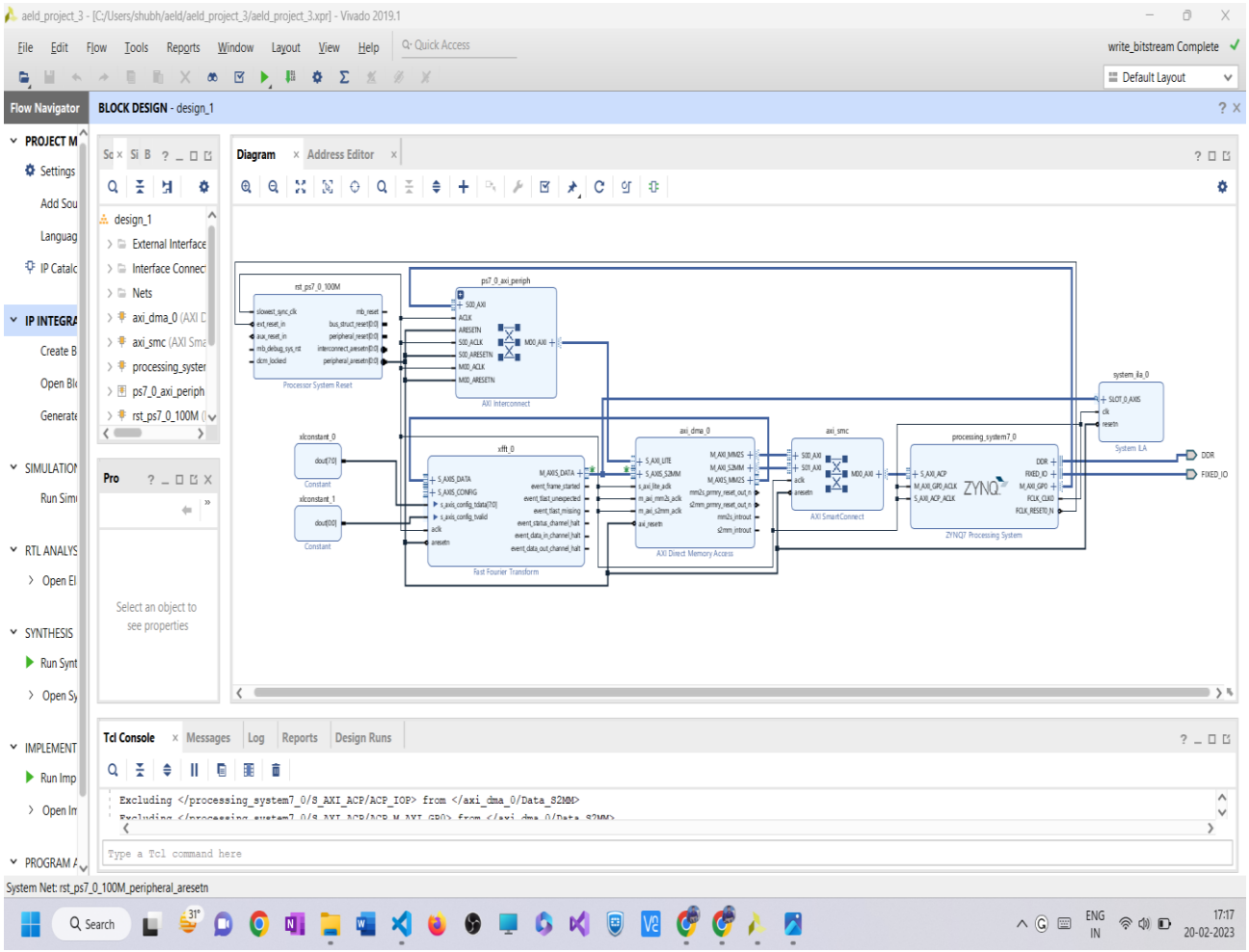


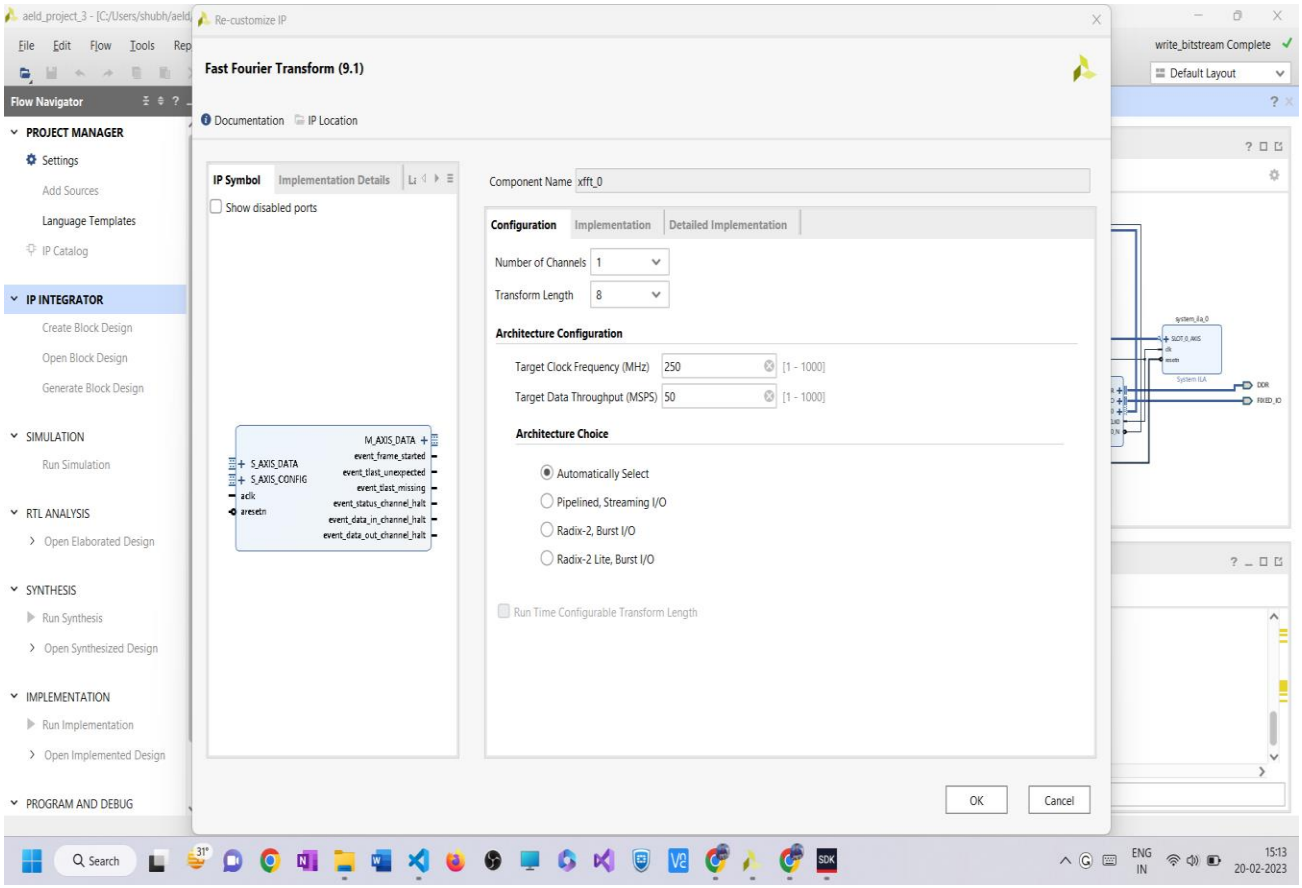
rand() *and* ***srand()*** *comparison*

Source: - Geeks for Geeks

For 8 FFT

BLOCK DIAGRAM





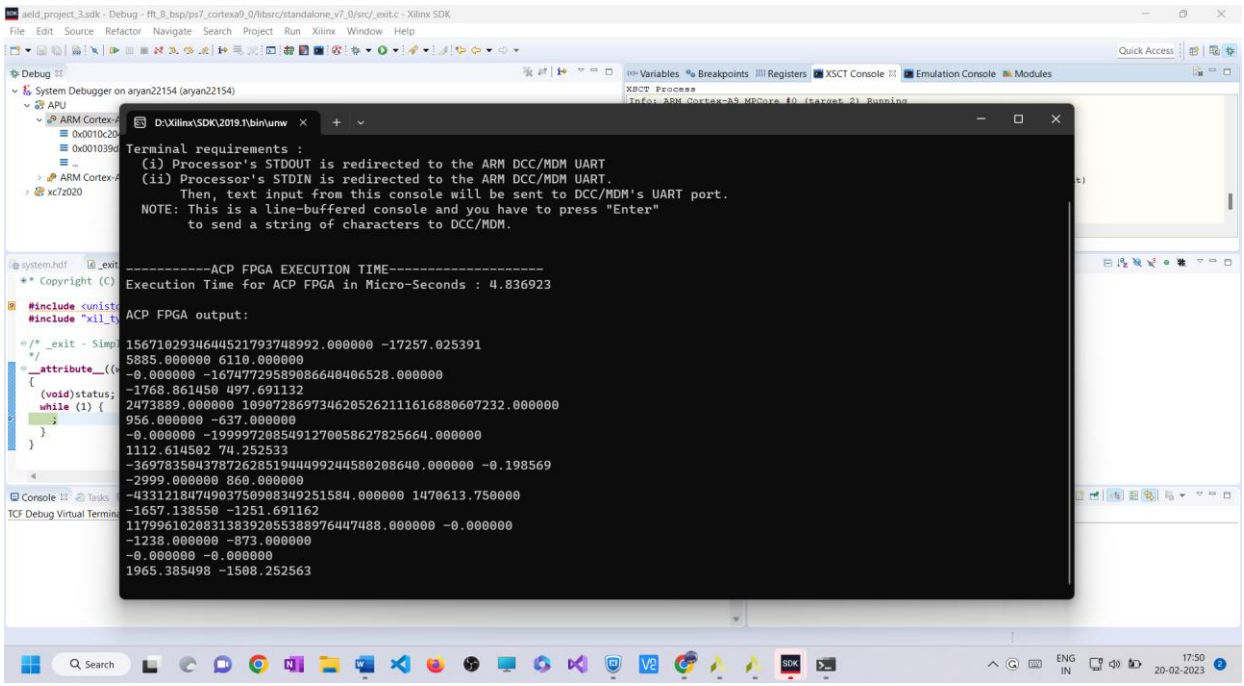
Code :-

```
#include <stdio.h>
#include <stdlib.h>
#include <complex.h>
#include <xtime_l.h>
#include <xparameters.h>
#include "xaxidma.h"
#define FFT_Size 8

#define XPAR_AXI_DMA_ACP_DEVICE_ID XPAR_AXI_DMA_0_DEVICE_ID
#define XPAR_AXI_DMA_ACP_BASEADDR XPAR_AXI_DMA_0_BASEADDR
int FFTPSVSACP()
{
    float complex FFT_input[FFT_Size];
    float complex FFT_PSoutput[FFT_Size];
    float complex FFT_rev[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;
      }
    XTime time_PS_start , time_PS_end;
    float complex FFT_ACPoutput[FFT_Size];
    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);
    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;
    }
    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_PSoutput[i]),
cimagf(FFT_PSoutput[i]));
            printf("%f %f\n", crealf(FFT_ACPoutput[i]),
cimagf(FFT_ACPoutput[i]));
        }
}
int main()
{
    init_platform();
    FFTPSVSACP();
    cleanup_platform();
}
```

JTAG TERMINAL: -



So, by using the hardware.

Execution Time -> 4.836923 Microseconds

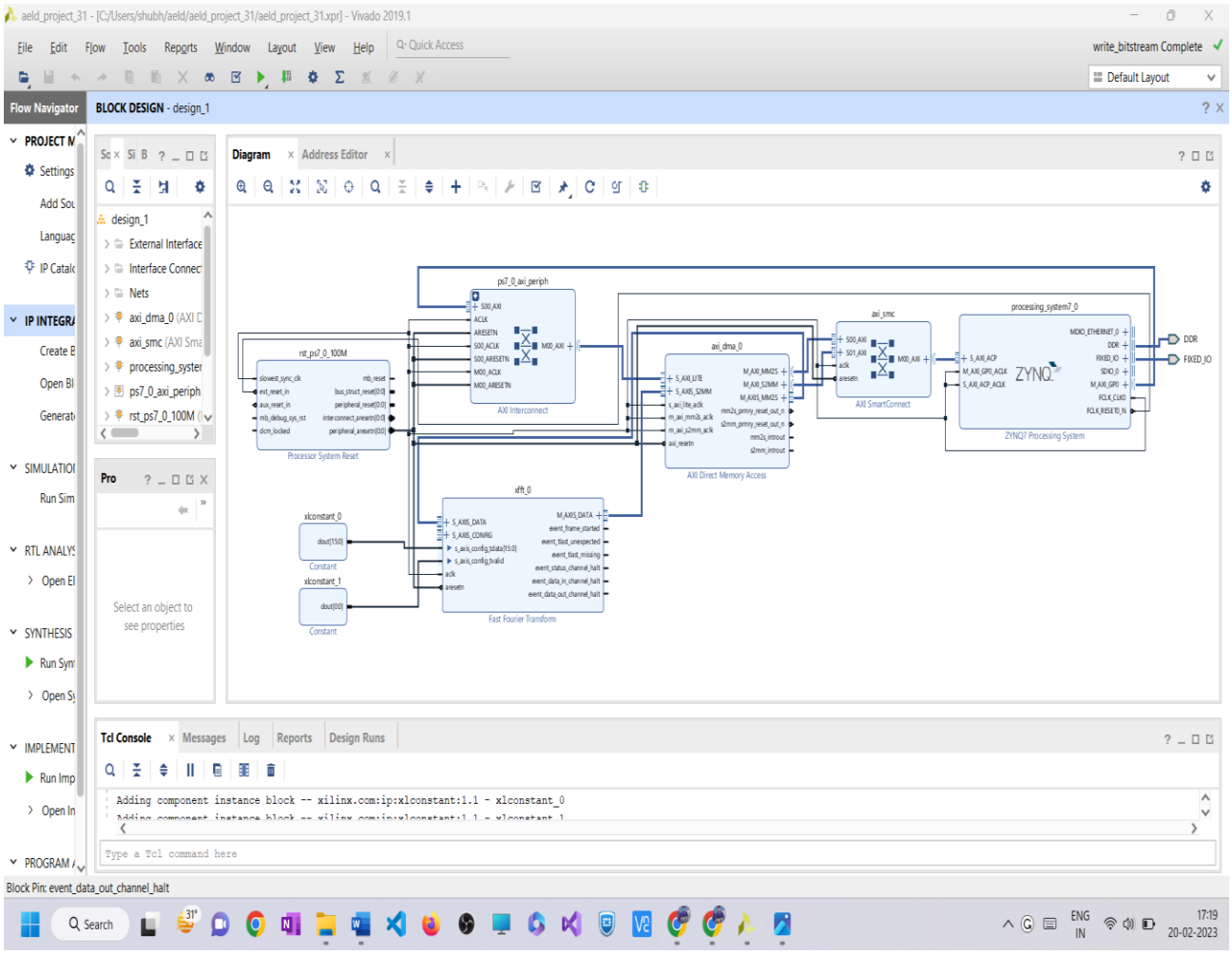
When we run using the software

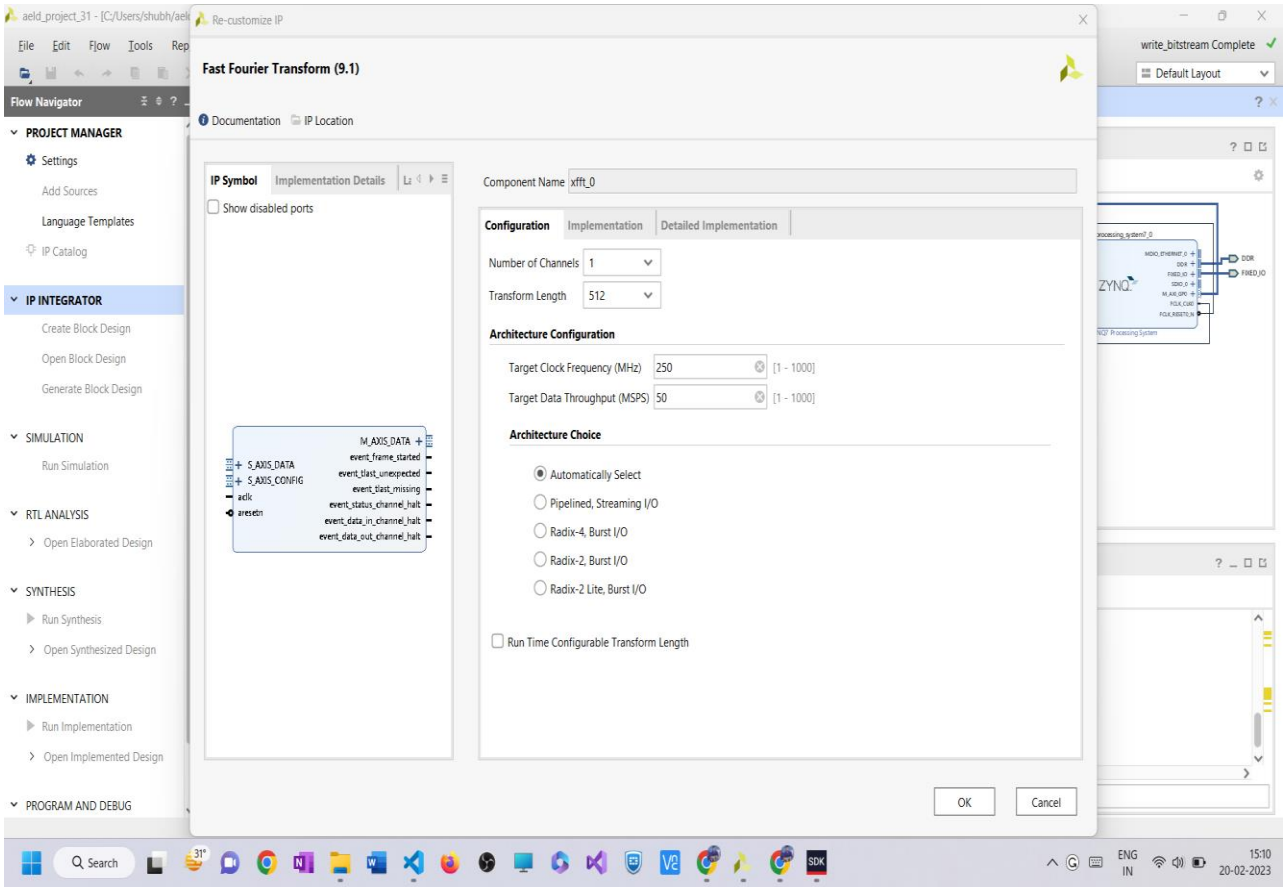
Execution Time -> 4.473846 Microseconds

Execution of the software looks the same, but now let’s come to the 512 Size FFT.

For 512 FFT

BLOCK DIAGRAM





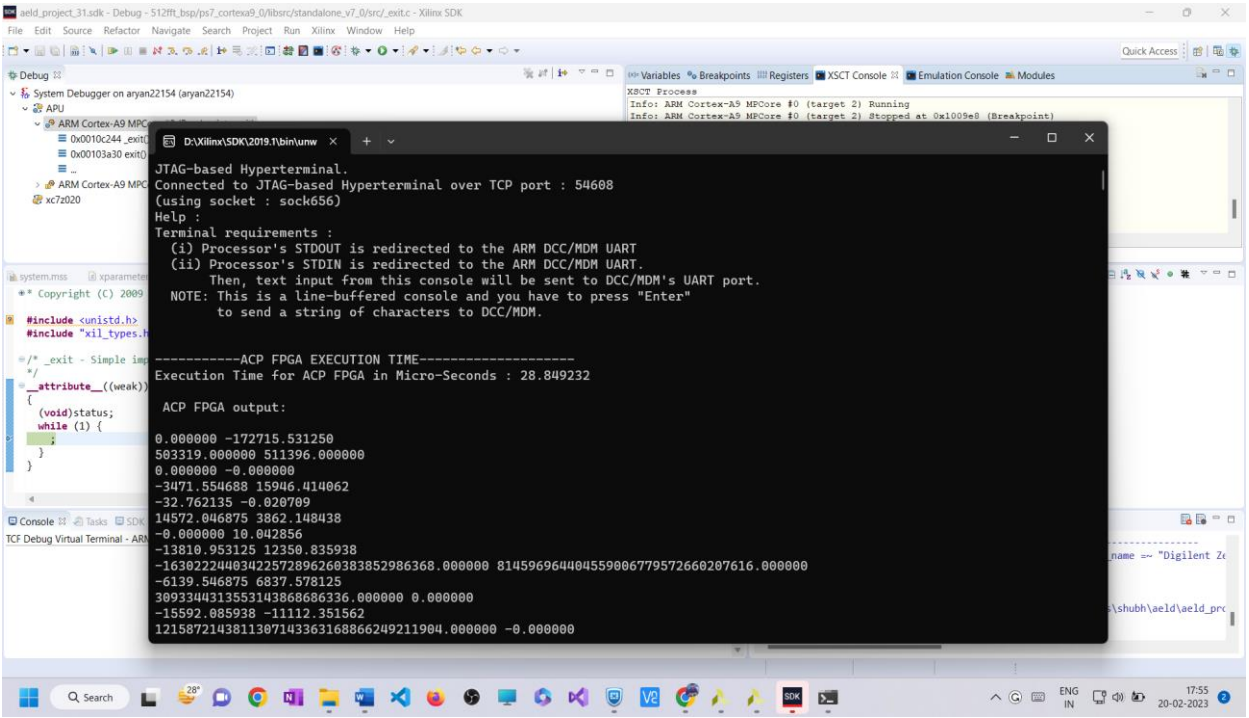
Code:-

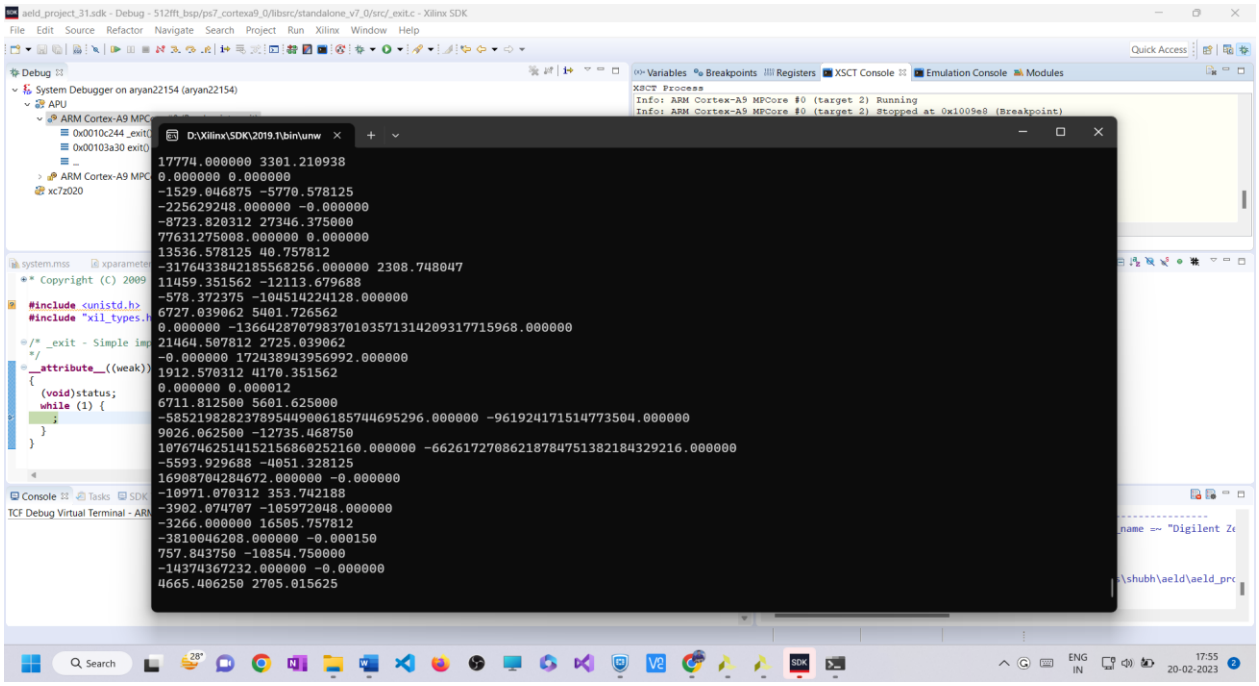
```
#include <stdio.h>
#include <stdlib.h>
#include <complex.h>
#include <xtime_l.h>
#include "platform.h"
#include "xil_printf.h"
#include "xparameters.h"
#include "xaxidma.h"
#define FFT_Size 512
#define XPAR_AXI_DMA_ACP_DEVICE_ID XPAR_AXI_DMA_0_DEVICE_ID
#define XPAR_AXI_DMA_ACP_BASEADDR XPAR_AXI_DMA_0_BASEADDR

int FFTPSVSACP()
{
    float complex FFT_input[FFT_Size];
    float complex FFT_PSoutput[FFT_Size];
    float complex FFT_rev[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;
    }
    XTime time_PS_start,time_PS_end;
    float complex FFT_ACPoutput[FFT_Size];
    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);
    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;
        }
    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_PSoutput[i]),
cimagf(FFT_PSoutput[i]));
            printf("%f %f\n", crealf(FFT_ACPoutput[i]),
cimagf(FFT_ACPoutput[i]));
        }
}
int main()
{
    init_platform();
    FFTPSVSACP();
    cleanup_platform();
    return 0;
}
```

JTAGTERMINAL





So, by using the hardware.

Execution Time -> 28.849232 Microseconds

When we run using the software

Execution Time -> 494.815399 Microseconds

So, in this, we can see that the hardware is much faster than the software.

Thank you.