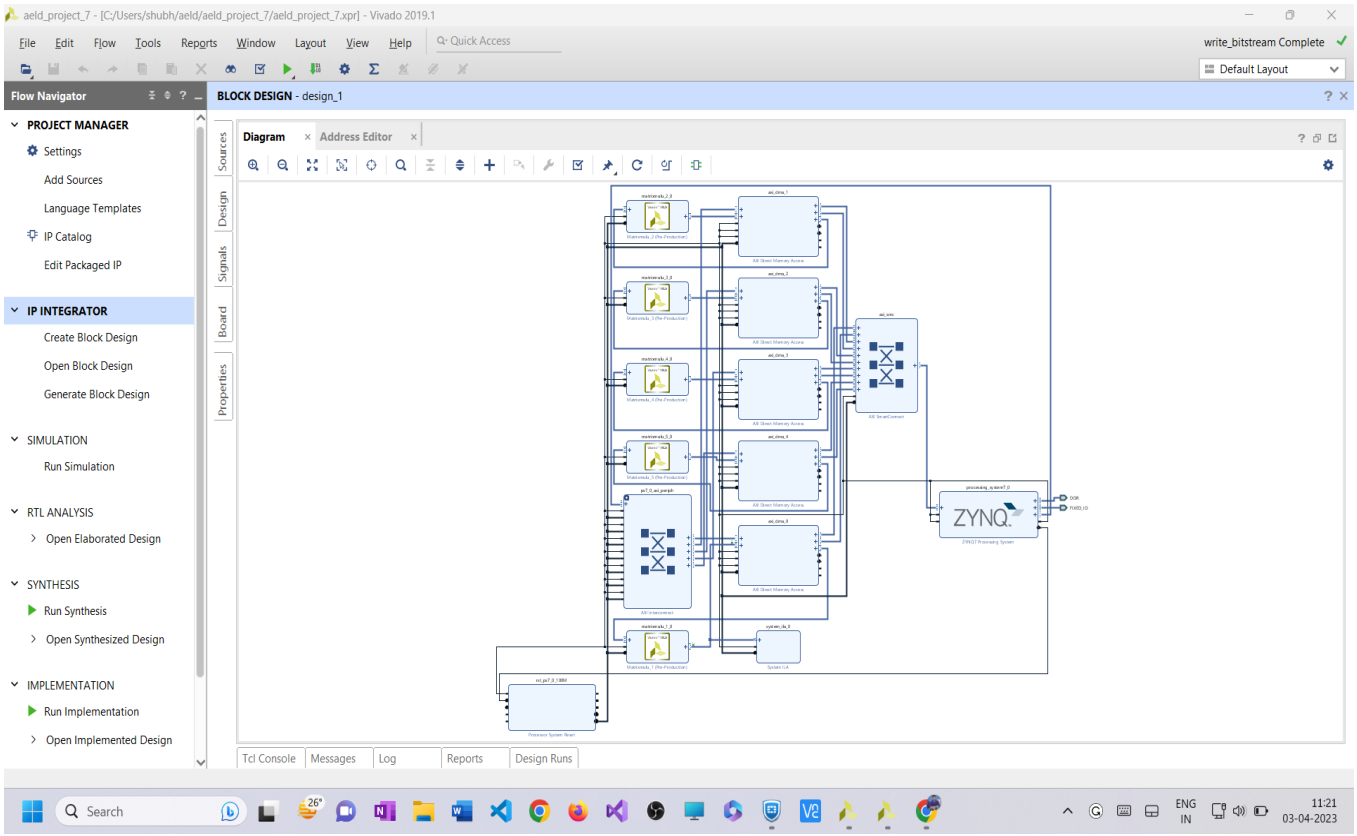
Name – Aryan Gupta

Roll No -MT22154

AELD Lab – 7 & 8

Block Diagram :-



Code: -

```
/*****************************************************************************
*
*
* Copyright (C) 2009 - 2014 Xilinx, Inc.  All rights reserved.
*
* Permission is hereby granted, free of charge, to any person obtaining a copy
* of this software and associated documentation files (the "Software"), to
deal
* in the Software without restriction, including without limitation the rights
* to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
* copies of the Software, and to permit persons to whom the Software is
* furnished to do so, subject to the following conditions:
*
* The above copyright notice and this permission notice shall be included in
* all copies or substantial portions of the Software.
*
* Use of the Software is limited solely to applications:
* (a) running on a Xilinx device, or
```

```
* (b) that interact with a Xilinx device through a bus or interconnect.
*
* THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
* IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
* FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL
* XILINX  BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY,
* WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF
* OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
* SOFTWARE.
*
* Except as contained in this notice, the name of the Xilinx shall not be used
* in advertising or otherwise to promote the sale, use or other dealings in
* this Software without prior written authorization from Xilinx.
*
******************************************************************************
/

/*
 * helloworld.c: simple test application
 *
 * This application configures UART 16550 to baud rate 9600.
 * PS7 UART (Zynq) is not initialized by this application, since
 * bootrom/bsp configures it to baud rate 115200
 *
 * ------------------------------------------------
 * | UART TYPE   BAUD RATE                        |
 * ------------------------------------------------
 *   uartns550   9600
 *   uartlite    Configurable only in HW design
 *   ps7_uart    115200 (configured by bootrom/bsp)
 */

#include <stdio.h>
#include <stdlib.h>
#include "xaxidma.h"
#include "xparameters.h"
#include "platform.h"
#include<xtime_l.h>
#define INP_SIZE 8
#define MATSIZE 8
float Find_input_A[8][8];
float Find_input_B[8][8];
float Find_outputps[8][8];
float Find_output1[8][8];
float Find_output2[8][8];
float Find_output3[8][8];
float Find_output4[8][8];
float Find_output5[8][8];
```

```
void input()
{
    printf("What is going on\n");
    for (int i=0;i<INP_SIZE;i++)
                        {
                            for(int j=0; j<INP_SIZE; j++)
                            {
                                Find_input_A[i][j]= (rand()%20);
                                Find_input_B[i][j]= (rand()%20);
                            }
                        }
}
void PS()
{
    float Find_outputps[INP_SIZE][INP_SIZE];
    XTime time_PS_start , time_PS_end;
    XTime_SetTime(0);
    XTime_GetTime(&time_PS_start);
    for(int i=0; i<MATSIZE; i++)
    {
        for(int j=0; j<MATSIZE; j++)
        {
            float res=0;
            for(int index=0; index<MATSIZE; index++)
            {
                res+=Find_input_A[i][index]*Find_input_B[index][j];
            }
            Find_outputps[i][j]=res;
        }
    }
    XTime_GetTime(&time_PS_end);
    printf("\n-----------PS FPGA EXECUTION TIME--------------------\n");
                                                    float time_PS = 0;
                                                    time_PS = (float)1.0 *
(time_PS_end - time_PS_start) / (COUNTS_PER_SECOND/1000000);
                                                    printf("Execution Time for
PS in Micro-Seconds : %f\n" , time_PS);
    for(int row=0; row<MATSIZE; row++)
        {
            for( int col=0; col<MATSIZE; col++)
            {
                printf("Input A %f, Input B
%f\n",Find_input_A[row][col],Find_input_B[row][col]);
            }
        }
    for(int row=0; row<MATSIZE; row++)
            {
                for(int col=0; col<MATSIZE; col++)
```

```
                {
                    printf("Output %f\n",Find_outputps[row][col]);
                }
            }
}
int Find_ACP1()
{
    float DMA_input[INP_SIZE*INP_SIZE*2];
    float Find_output_DMA[INP_SIZE*INP_SIZE];
    int status;
     XAxiDma_Config *DMA_confptracp; //DMA configuration pointer
     XAxiDma AxiDMAacp; // DMA instance pointer
     DMA_confptracp = XAxiDma_LookupConfig(XPAR_AXI_DMA_0_DEVICE_ID);
     status = XAxiDma_CfgInitialize(&AxiDMAacp, DMA_confptracp);
     if(status != XST_SUCCESS)
     {
        printf("ACP DMA Init Failed\t\n");
        return XST_FAILURE;
     }
     XTime time_ACP_start , time_ACP_end;
                 XTime_SetTime(0);
                 XTime_GetTime(&time_ACP_start);
            int index=0;
            for(int row=0; row<MATSIZE; row++)
            {
                for(int col=0; col<MATSIZE; col++)
                {
                    DMA_input[index]=Find_input_A[row][col];
                    index++;
                }
            }
            for(int row=0; row<MATSIZE; row++)
            {
                for(int col=0; col<MATSIZE; col++)
                {
                    DMA_input[index]=Find_input_B[row][col];
                    index++;
                }
            }
//          for(int i=0; i<INP_SIZE*INP_SIZE*2; i++)
//          {
//              printf("Input %f \n ",DMA_input[i]);
//          }
               status = XAxiDma_SimpleTransfer(&AxiDMAacp,
(UINTPTR)Find_output_DMA,(sizeof(float)*INP_SIZE*INP_SIZE),XAXIDMA_DEVICE_TO_D
MA);
                status = XAxiDma_SimpleTransfer(&AxiDMAacp, (UINTPTR)DMA_input,
(sizeof(float)*INP_SIZE*INP_SIZE*2),XAXIDMA_DMA_TO_DEVICE);
```

```
               status = XAxiDma_ReadReg(XPAR_AXI_DMA_0_BASEADDR,0x04) &
0x00000002;
               while(status!=0x00000002)
               {
                 status = XAxiDma_ReadReg(XPAR_AXI_DMA_0_BASEADDR,0x04) &
0x00000002;
               }
               status = XAxiDma_ReadReg(XPAR_AXI_DMA_0_BASEADDR,0x34) &
0x00000002;

              while(status!=0x00000002)
              {
                status = XAxiDma_ReadReg(XPAR_AXI_DMA_0_BASEADDR,0x34) &
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);
                                for(int i = 0 ; i<INP_SIZE*INP_SIZE; i++)
                                {
                                    printf("Output  :
%f\n",Find_output_DMA[i]);
                                }
                                index=0;
                           for(int row=0; row<MATSIZE; row++)
                                    {
                                        for(int col=0; col<MATSIZE; col++)
                                        {
                                            Find_output1[row][col]=Find_ou
tput_DMA[index];

                                            index++;
                                        }
                                    }
     return 0;
}
int Find_ACP2()
{
    float DMA_input[INP_SIZE*INP_SIZE*2];
    float Find_output_DMA[INP_SIZE*INP_SIZE];
    int status;
     XAxiDma_Config *DMA_confptracp; //DMA configuration pointer
     XAxiDma AxiDMAacp; // DMA instance pointer
     DMA_confptracp = XAxiDma_LookupConfig(XPAR_AXI_DMA_1_DEVICE_ID);
```

```
    status = XAxiDma_CfgInitialize(&AxiDMAacp, DMA_confptracp);
    if(status != XST_SUCCESS)
    {
       printf("ACP DMA Init Failed\t\n");
       return XST_FAILURE;
    }
    XTime time_ACP_start , time_ACP_end;
                XTime_SetTime(0);
                XTime_GetTime(&time_ACP_start);
            int index=0;
            for(int row=0; row<MATSIZE; row++)
            {
                for(int col=0; col<MATSIZE; col++)
                {
                    DMA_input[index]=Find_input_A[row][col];
                    index++;
                }
            }
            for(int row=0; row<MATSIZE; row++)
            {
                for(int col=0; col<MATSIZE; col++)
                {
                    DMA_input[index]=Find_input_B[row][col];
                    index++;
                }
            }
//          for(int i=0; i<INP_SIZE*INP_SIZE*2; i++)
//          {
//              printf("Input %f \n ",DMA_input[i]);
//          }
               status = XAxiDma_SimpleTransfer(&AxiDMAacp,
(UINTPTR)Find_output_DMA,(sizeof(float)*INP_SIZE*INP_SIZE),XAXIDMA_DEVICE_TO_D
MA);
               status = XAxiDma_SimpleTransfer(&AxiDMAacp, (UINTPTR)DMA_input,
(sizeof(float)*INP_SIZE*INP_SIZE*2),XAXIDMA_DMA_TO_DEVICE);
               status = XAxiDma_ReadReg(XPAR_AXI_DMA_0_BASEADDR,0x04) &
0x00000002;
               while(status!=0x00000002)
               {
                 status = XAxiDma_ReadReg(XPAR_AXI_DMA_0_BASEADDR,0x04) &
0x00000002;
               }
               status = XAxiDma_ReadReg(XPAR_AXI_DMA_0_BASEADDR,0x34) &
0x00000002;

              while(status!=0x00000002)
              {
```

```
                status = XAxiDma_ReadReg(XPAR_AXI_DMA_0_BASEADDR,0x34) &
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);
                                  for(int i = 0 ; i<INP_SIZE*INP_SIZE; i++)
                                  {
                                          printf("Output  :
%f\n",Find_output_DMA[i]);
                                  }
                                  index=0;
                             for(int row=0; row<MATSIZE; row++)
                                        {
                                            for(int col=0; col<MATSIZE; col++)
                                            {
                                                Find_output2[row][col]=Find_ou
tput_DMA[index];

                                                index++;
                                            }
                                        }
     return 0;
}
int Find_ACP3()
{
    float DMA_input[INP_SIZE*INP_SIZE*2];
    float Find_output_DMA[INP_SIZE*INP_SIZE];
    int status;
     XAxiDma_Config *DMA_confptracp; //DMA configuration pointer
     XAxiDma AxiDMAacp; // DMA instance pointer
     DMA_confptracp = XAxiDma_LookupConfig(XPAR_AXI_DMA_2_DEVICE_ID);
     status = XAxiDma_CfgInitialize(&AxiDMAacp, DMA_confptracp);
     if(status != XST_SUCCESS)
     {
        printf("ACP DMA Init Failed\t\n");
        return XST_FAILURE;
     }
     XTime time_ACP_start , time_ACP_end;
                 XTime_SetTime(0);
                 XTime_GetTime(&time_ACP_start);
            int index=0;
            for(int row=0; row<MATSIZE; row++)
            {
```

```
                for(int col=0; col<MATSIZE; col++)
                {
                    DMA_input[index]=Find_input_A[row][col];
                    index++;
                }
            }
            for(int row=0; row<MATSIZE; row++)
            {
                for(int col=0; col<MATSIZE; col++)
                {
                    DMA_input[index]=Find_input_B[row][col];
                    index++;
                }
            }
//          for(int i=0; i<INP_SIZE*INP_SIZE*2; i++)
//          {
//              printf("Input %f \n ",DMA_input[i]);
//          }
               status = XAxiDma_SimpleTransfer(&AxiDMAacp,
(UINTPTR)Find_output_DMA,(sizeof(float)*INP_SIZE*INP_SIZE),XAXIDMA_DEVICE_TO_D
MA);
                status = XAxiDma_SimpleTransfer(&AxiDMAacp, (UINTPTR)DMA_input,
(sizeof(float)*INP_SIZE*INP_SIZE*2),XAXIDMA_DMA_TO_DEVICE);
                status = XAxiDma_ReadReg(XPAR_AXI_DMA_0_BASEADDR,0x04) &
0x00000002;
                while(status!=0x00000002)
                {
                  status = XAxiDma_ReadReg(XPAR_AXI_DMA_0_BASEADDR,0x04) &
0x00000002;
                }
                status = XAxiDma_ReadReg(XPAR_AXI_DMA_0_BASEADDR,0x34) &
0x00000002;

               while(status!=0x00000002)
               {
                 status = XAxiDma_ReadReg(XPAR_AXI_DMA_0_BASEADDR,0x34) &
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);
                                    for(int i = 0 ; i<INP_SIZE*INP_SIZE; i++)
                                    {
```

```
                                        printf("Output   :
%f\n",Find_output_DMA[i]);
                                }
                                index=0;
                            for(int row=0; row<MATSIZE; row++)
                                    {
                                        for(int col=0; col<MATSIZE; col++)
                                        {
                                            Find_output3[row][col]=Find_ou
tput_DMA[index];
                                            index++;
                                        }
                                    }
     return 0;
}
int Find_ACP4()
{
    float DMA_input[INP_SIZE*INP_SIZE*2];
    float Find_output_DMA[INP_SIZE*INP_SIZE];
    int status;
     XAxiDma_Config *DMA_confptracp; //DMA configuration pointer
     XAxiDma AxiDMAacp; // DMA instance pointer
     DMA_confptracp = XAxiDma_LookupConfig(XPAR_AXI_DMA_3_DEVICE_ID);
     status = XAxiDma_CfgInitialize(&AxiDMAacp, DMA_confptracp);
     if(status != XST_SUCCESS)
     {
        printf("ACP DMA Init Failed\t\n");
        return XST_FAILURE;
     }
     XTime time_ACP_start , time_ACP_end;
                 XTime_SetTime(0);
                 XTime_GetTime(&time_ACP_start);
            int index=0;
            for(int row=0; row<MATSIZE; row++)
            {
                for(int col=0; col<MATSIZE; col++)
                {
                    DMA_input[index]=Find_input_A[row][col];
                    index++;
                }
            }
            for(int row=0; row<MATSIZE; row++)
            {
                for(int col=0; col<MATSIZE; col++)
                {
                    DMA_input[index]=Find_input_B[row][col];
                    index++;
                }
```

```
            }
//          for(int i=0; i<INP_SIZE*INP_SIZE*2; i++)
//          {
//              printf("Input %f \n ",DMA_input[i]);
//          }
               status = XAxiDma_SimpleTransfer(&AxiDMAacp,
(UINTPTR)Find_output_DMA,(sizeof(float)*INP_SIZE*INP_SIZE),XAXIDMA_DEVICE_TO_D
MA);
               status = XAxiDma_SimpleTransfer(&AxiDMAacp, (UINTPTR)DMA_input,
(sizeof(float)*INP_SIZE*INP_SIZE*2),XAXIDMA_DMA_TO_DEVICE);
               status = XAxiDma_ReadReg(XPAR_AXI_DMA_0_BASEADDR,0x04) &
0x00000002;
               while(status!=0x00000002)
               {
                 status = XAxiDma_ReadReg(XPAR_AXI_DMA_0_BASEADDR,0x04) &
0x00000002;
               }
               status = XAxiDma_ReadReg(XPAR_AXI_DMA_0_BASEADDR,0x34) &
0x00000002;

              while(status!=0x00000002)
              {
                status = XAxiDma_ReadReg(XPAR_AXI_DMA_0_BASEADDR,0x34) &
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);
                                  for(int i = 0 ; i<INP_SIZE*INP_SIZE; i++)
                                  {
                                          printf("Output  :
%f\n",Find_output_DMA[i]);
                                  }
                                  index=0;
                             for(int row=0; row<MATSIZE; row++)
                                        {
                                           for(int col=0; col<MATSIZE; col++)
                                           {
                                               Find_output4[row][col]=Find_ou
tput_DMA[index];

                                               index++;
                                           }
                                        }
```

```
     return 0;
}
int Find_ACP5()
{
    float DMA_input[INP_SIZE*INP_SIZE*2];
    float Find_output_DMA[INP_SIZE*INP_SIZE];
    int status;
     XAxiDma_Config *DMA_confptracp; //DMA configuration pointer
     XAxiDma AxiDMAacp; // DMA instance pointer
     DMA_confptracp = XAxiDma_LookupConfig(XPAR_AXI_DMA_4_DEVICE_ID);
     status = XAxiDma_CfgInitialize(&AxiDMAacp, DMA_confptracp);
     if(status != XST_SUCCESS)
     {
        printf("ACP DMA Init Failed\t\n");
        return XST_FAILURE;
     }
     XTime time_ACP_start , time_ACP_end;
                 XTime_SetTime(0);
                 XTime_GetTime(&time_ACP_start);
            int index=0;
            for(int row=0; row<MATSIZE; row++)
            {
                for(int col=0; col<MATSIZE; col++)
                {
                    DMA_input[index]=Find_input_A[row][col];
                    index++;
                }
            }
            for(int row=0; row<MATSIZE; row++)
            {
                for(int col=0; col<MATSIZE; col++)
                {
                    DMA_input[index]=Find_input_B[row][col];
                    index++;
                }
            }
//          for(int i=0; i<INP_SIZE*INP_SIZE*2; i++)
//          {
//              printf("Input %f \n ",DMA_input[i]);
//          }
               status = XAxiDma_SimpleTransfer(&AxiDMAacp,
(UINTPTR)Find_output_DMA,(sizeof(float)*INP_SIZE*INP_SIZE),XAXIDMA_DEVICE_TO_D
MA);
               status = XAxiDma_SimpleTransfer(&AxiDMAacp, (UINTPTR)DMA_input,
(sizeof(float)*INP_SIZE*INP_SIZE*2),XAXIDMA_DMA_TO_DEVICE);
               status = XAxiDma_ReadReg(XPAR_AXI_DMA_0_BASEADDR,0x04) &
0x00000002;
               while(status!=0x00000002)
```

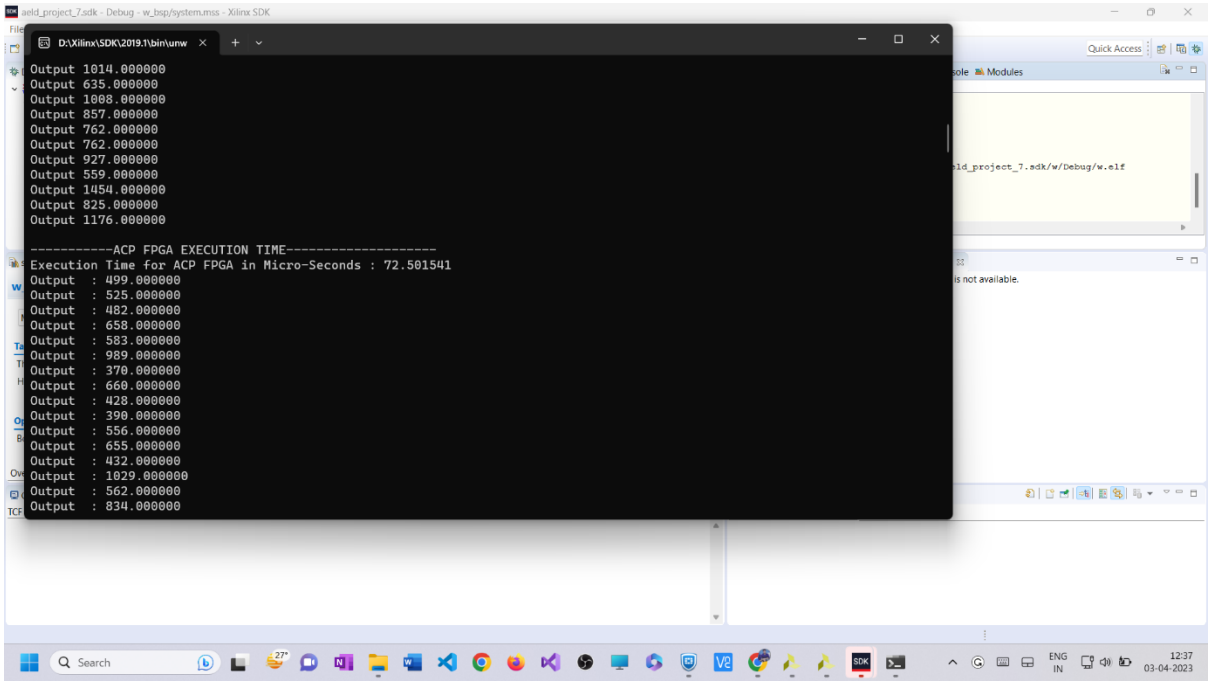
```
               {
                 status = XAxiDma_ReadReg(XPAR_AXI_DMA_0_BASEADDR,0x04) &
0x00000002;
               }
               status = XAxiDma_ReadReg(XPAR_AXI_DMA_0_BASEADDR,0x34) &
0x00000002;

             while(status!=0x00000002)
             {
               status = XAxiDma_ReadReg(XPAR_AXI_DMA_0_BASEADDR,0x34) &
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);
                               for(int i = 0 ; i<INP_SIZE*INP_SIZE; i++)
                               {
                                       printf("Output  :
%f\n",Find_output_DMA[i]);
                               }
                               index=0;
                          for(int row=0; row<MATSIZE; row++)
                                     {
                                        for(int col=0; col<MATSIZE; col++)
                                        {
                                            Find_output5[row][col]=Find_ou
tput_DMA[index];

                                            index++;
                                        }
                                     }
     return 0;
}
int compare1()
{
    for(int i=0; i<MATSIZE; i++)
    {
        for(int j=0; j<MATSIZE; j++)
        {
            if(Find_output1[i][j]!=Find_outputps[i][j])
            {
                return 0;
            }
        }
```

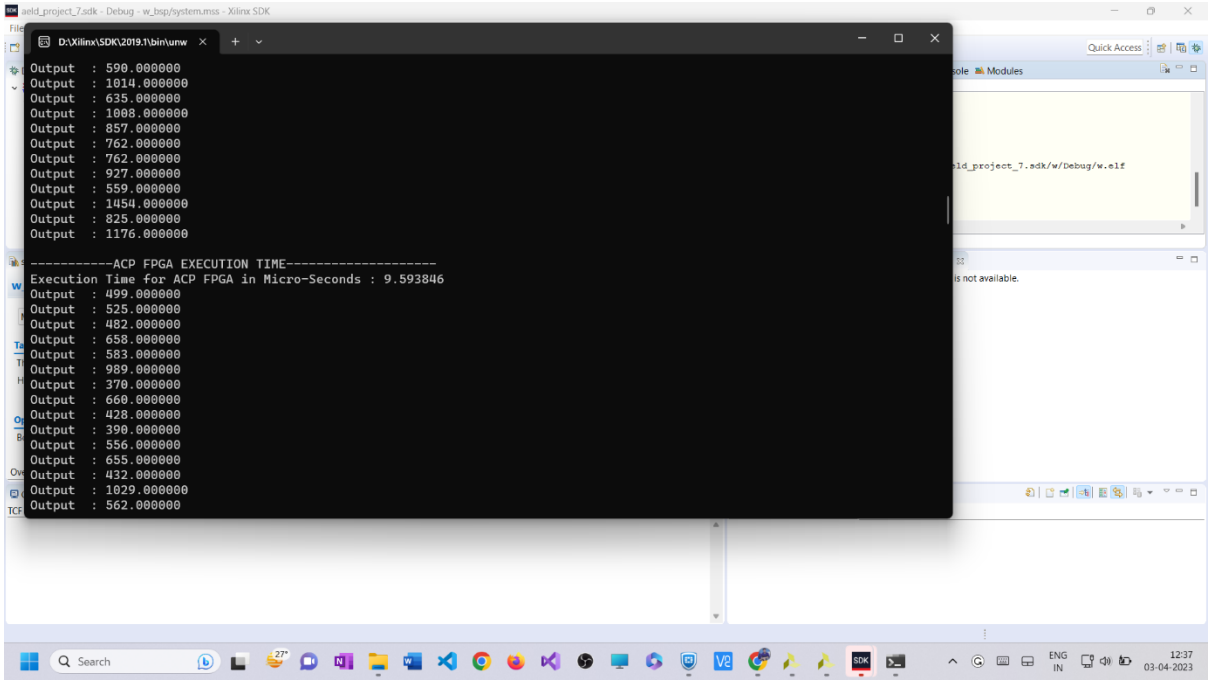
```
    }
    return 1;
}
int compare2()
{
    for(int i=0; i<MATSIZE; i++)
    {
        for(int j=0; j<MATSIZE; j++)
        {
            if(Find_output2[i][j]!=Find_outputps[i][j])
            {
                return 0;
            }
        }
    }
    return 1;
}
int compare3()
{
    for(int i=0; i<MATSIZE; i++)
    {
        for(int j=0; j<MATSIZE; j++)
        {
            if(Find_output3[i][j]!=Find_outputps[i][j])
            {
                return 0;
            }
        }
    }
    return 1;
}
int compare4()
{
    for(int i=0; i<MATSIZE; i++)
    {
        for(int j=0; j<MATSIZE; j++)
        {
            if(Find_output4[i][j]!=Find_outputps[i][j])
            {
                return 0;
            }
        }
    }
    return 1;
}
int compare5()
{
    for(int i=0; i<MATSIZE; i++)
```

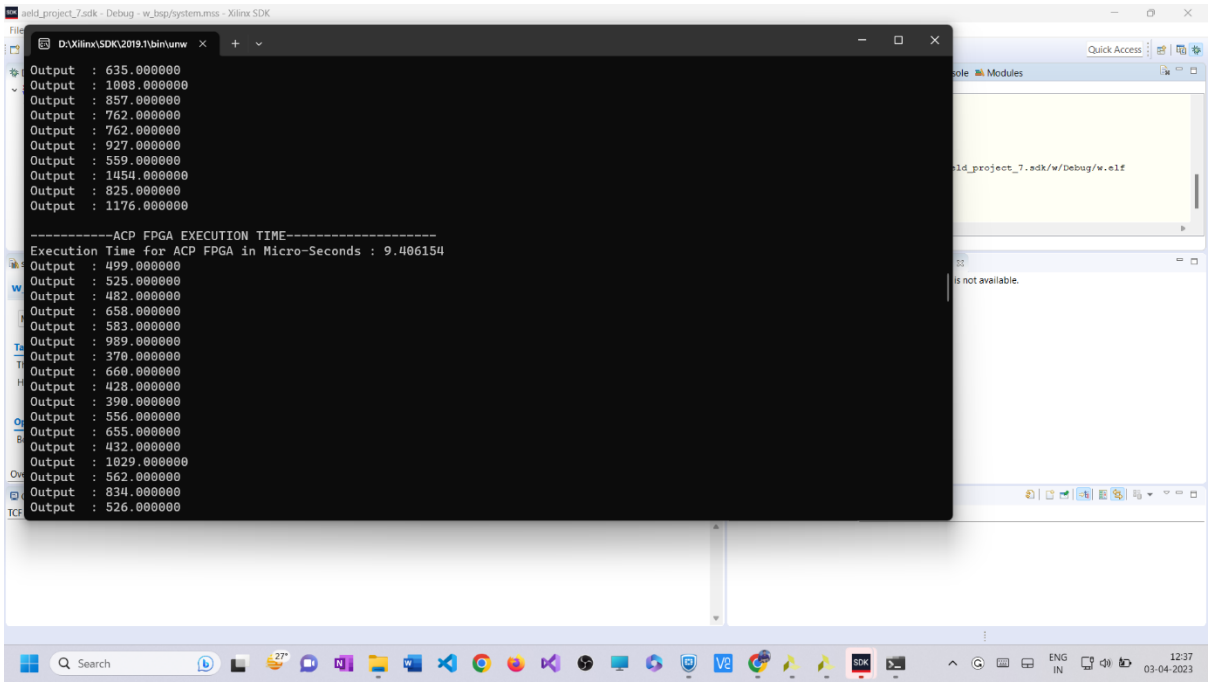
```
    {
        for(int j=0; j<MATSIZE; j++)
        {
            if(Find_output5[i][j]!=Find_outputps[i][j])
            {
                return 0;
            }
        }
    }
    return 1;
}
int main()
{
    init_platform();
    input();
    PS();
    Find_ACP1();
    Find_ACP2();
    Find_ACP3();
    Find_ACP4();
    Find_ACP5();
    int ans=compare1();
    int ans1=compare2();
    int ans2=compare3();
    int ans3=compare4();
    int ans4=compare5();
    if(ans==1)
    {
        printf("You are great\n");
    }
    else
    {
        printf("No good at all\n");
    }
    if(ans1==1)
    {
        printf("You are great\n");
    }
    else
    {
        printf("No good at all");
    }
    if(ans2==1)
    {
        printf("You are great\n");
    }
    else
    {
```

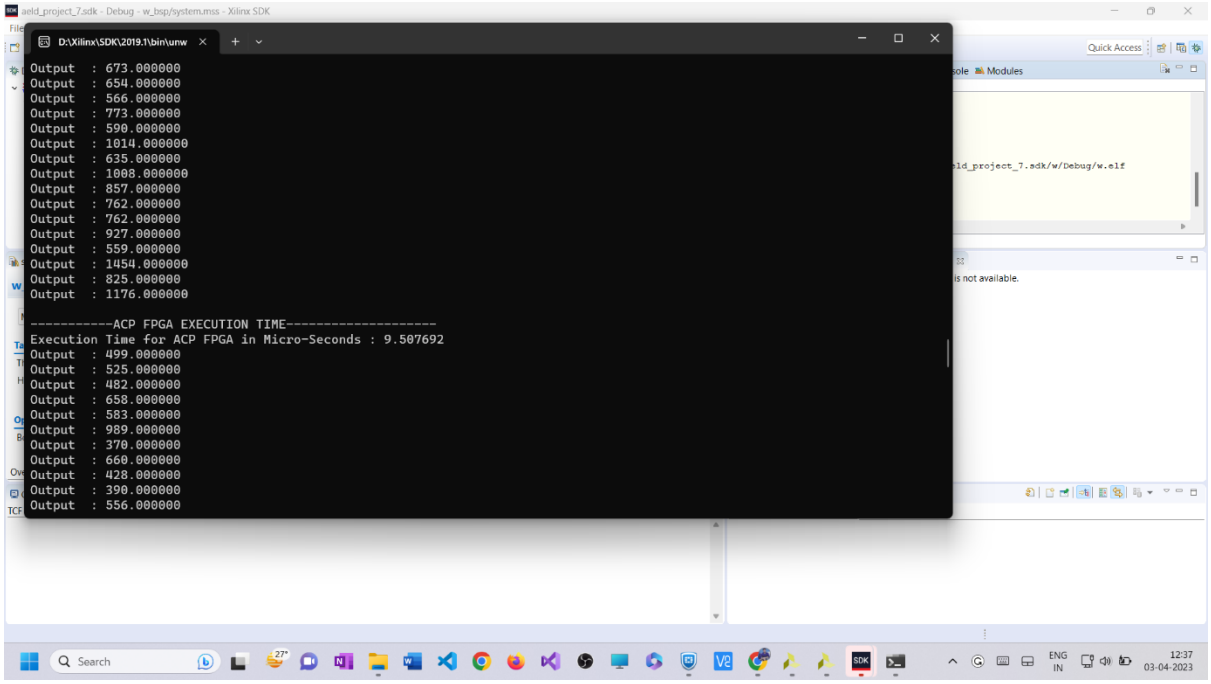
```
        printf("No good at all\n");
    }
    if(ans3==1)
    {
        printf("You are great\n");
    }
    else
    {
        printf("No good at all\n");
    }
    if(ans4==1)
    {
        printf("You are great\n");
    }
    else
    {
        printf("No good at all");
    }
    cleanup_platform();
    return 0;
}
```

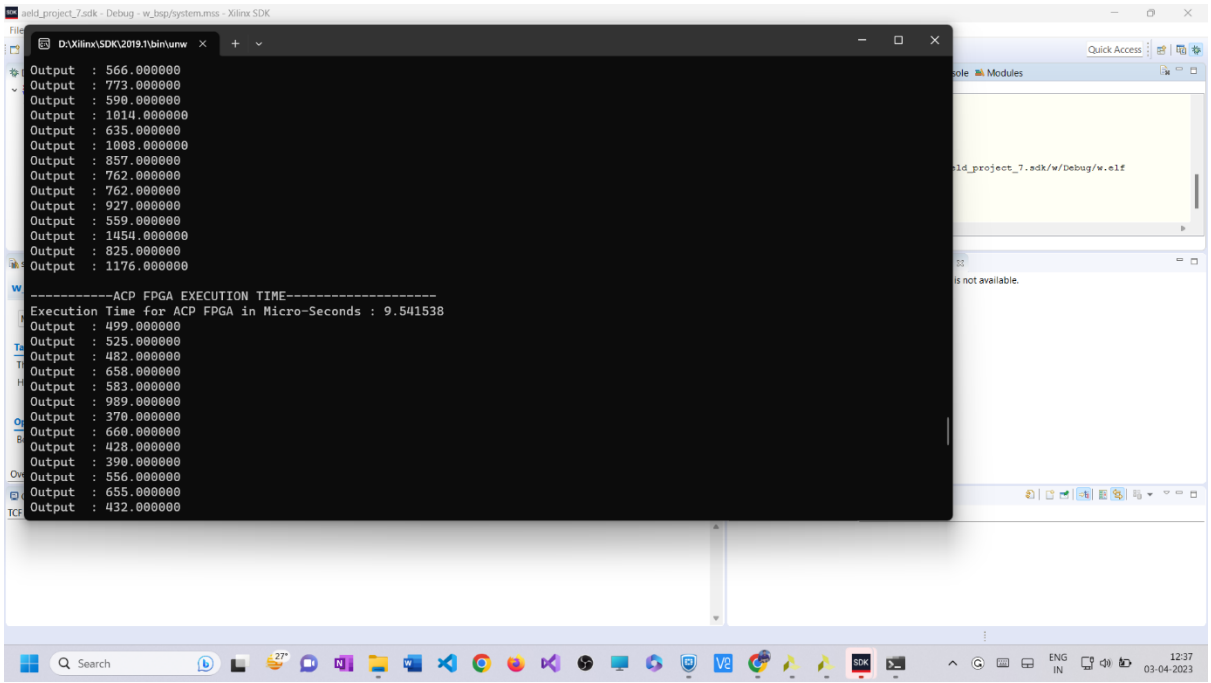
Jtagterminal:-



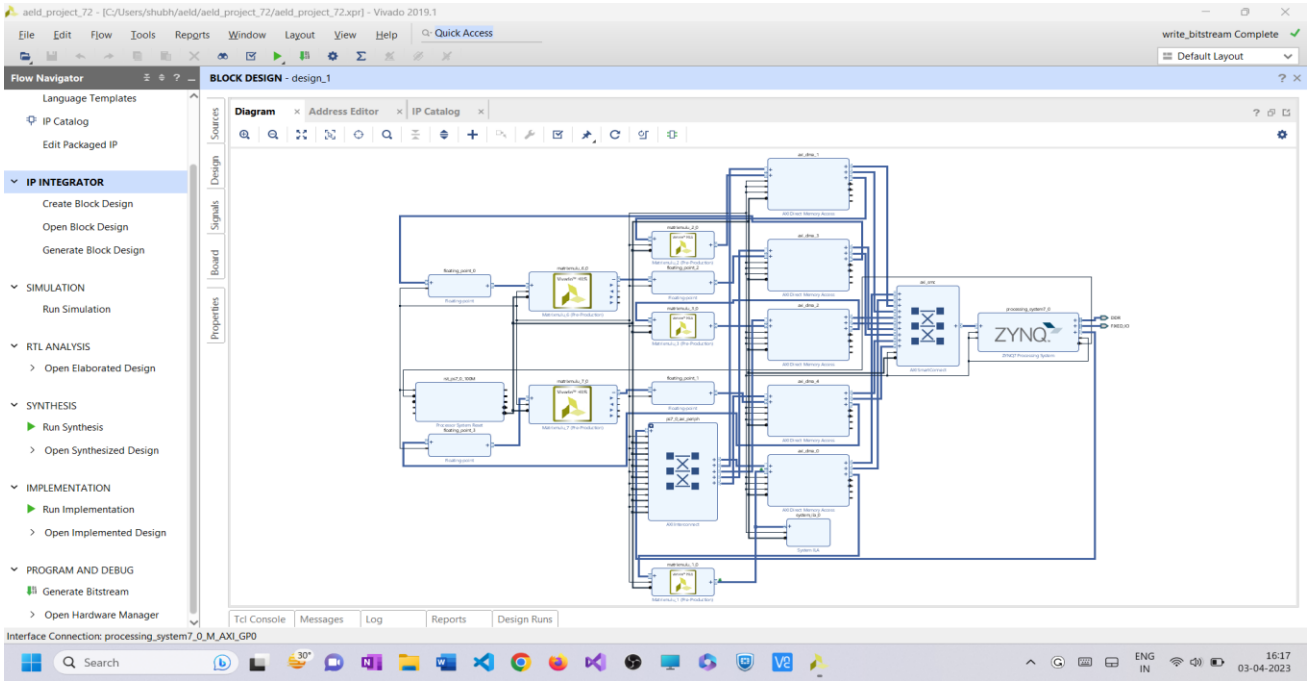

D:\Xilinx\SDK\2019.1\bin\unw
Output : 590.000000
Output : 1014.000000
Output : 635.000000
Output : 1008.000000
Output : 857.000000
Output : 762.000000
Output : 762.000000
Output : 927.000000
Output : 559.000000
Output : 1454.000000
Output : 825.000000
Output : 1176.000000
----------ACP FPGA EXECUTION TIME--------------------
Execution Time for ACP FPGA in Micro-Seconds : 9.593846
Output : 499.000000
Output : 525.000000
Output : 482.000000
Output : 658.000000
Output : 583.000000
Output : 989.000000
Output : 370.000000
Output : 660.000000
Output : 428.000000
Output : 390.000000
Output : 556.000000
Output : 655.000000
Output : 432.000000
Output : 1029.000000
Output : 562.000000


D:\Xilinx\SDK\2019.1\bin\unw
Output : 635.000000
Output : 1008.000000
Output : 857.000000
Output : 762.000000
Output : 762.000000
Output : 927.000000
Output : 559.000000
Output : 1454.000000
Output : 825.000000
Output : 1176.000000
----------ACP FPGA EXECUTION TIME--------------------
Execution Time for ACP FPGA in Micro-Seconds : 9.406154
Output : 499.000000
Output : 525.000000
Output : 482.000000
Output : 658.000000
Output : 583.000000
Output : 989.000000
Output : 370.000000
Output : 660.000000
Output : 428.000000
Output : 390.000000
Output : 556.000000
Output : 655.000000
Output : 432.000000
Output : 1029.000000
Output : 562.000000
Output : 834.000000
Output : 526.000000


D:\Xilinx\SDK\2019.1\bin\unw
Output : 673.000000
Output : 654.000000
Output : 566.000000
Output : 773.000000
Output : 590.000000
Output : 1014.000000
Output : 635.000000
Output : 1008.000000
Output : 857.000000
Output : 762.000000
Output : 762.000000
Output : 927.000000
Output : 559.000000
Output : 1454.000000
Output : 825.000000
Output : 1176.000000
----------ACP FPGA EXECUTION TIME--------------------
Execution Time for ACP FPGA in Micro-Seconds : 9.507692
Output : 499.000000
Output : 525.000000
Output : 482.000000
Output : 658.000000
Output : 583.000000
Output : 989.000000
Output : 370.000000
Output : 660.000000
Output : 428.000000
Output : 390.000000
Output : 556.000000


D:\Xilinx\SDK\2019.1\bin\unw
Output : 566.000000
Output : 773.000000
Output : 590.000000
Output : 1014.000000
Output : 635.000000
Output : 1008.000000
Output : 857.000000
Output : 762.000000
Output : 762.000000
Output : 927.000000
Output : 559.000000
Output : 1454.000000
Output : 825.000000
Output : 1176.000000
----------ACP FPGA EXECUTION TIME--------------------
Execution Time for ACP FPGA in Micro-Seconds : 9.541538
Output : 499.000000
Output : 525.000000
Output : 482.000000
Output : 658.000000
Output : 583.000000
Output : 989.000000
Output : 370.000000
Output : 660.000000
Output : 428.000000
Output : 390.000000
Output : 556.000000
Output : 655.000000
Output : 432.000000

Block Diagram:-



Code: -

```
/*****************************************************************************
*
*
* Copyright (C) 2009 - 2014 Xilinx, Inc.  All rights reserved.
*
* Permission is hereby granted, free of charge, to any person obtaining a copy
* of this software and associated documentation files (the "Software"), to
deal
* in the Software without restriction, including without limitation the rights
* to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
* copies of the Software, and to permit persons to whom the Software is
* furnished to do so, subject to the following conditions:
*
* The above copyright notice and this permission notice shall be included in
* all copies or substantial portions of the Software.
*
* Use of the Software is limited solely to applications:
* (a) running on a Xilinx device, or
* (b) that interact with a Xilinx device through a bus or interconnect.
*
* THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
* IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
* FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL
* XILINX  BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY,
* WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF
* OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
* SOFTWARE.
```

```
*
* Except as contained in this notice, the name of the Xilinx shall not be used
* in advertising or otherwise to promote the sale, use or other dealings in
* this Software without prior written authorization from Xilinx.
*
******************************************************************************
/

/*
 * helloworld.c: simple test application
 *
 * This application configures UART 16550 to baud rate 9600.
 * PS7 UART (Zynq) is not initialized by this application, since
 * bootrom/bsp configures it to baud rate 115200
 *
 * ------------------------------------------------
 * | UART TYPE   BAUD RATE                        |
 * ------------------------------------------------
 *   uartns550   9600
 *   uartlite    Configurable only in HW design
 *   ps7_uart    115200 (configured by bootrom/bsp)
 */

#include <stdio.h>
#include <stdlib.h>
#include "xaxidma.h"
#include "xparameters.h"
#include "platform.h"
#include<xtime_l.h>
#define INP_SIZE 8
#define MATSIZE 8
float Find_input_A[8][8];
float Find_input_B[8][8];
float Find_outputps[8][8];
float Find_output1[8][8];
float Find_output2[8][8];
float Find_output3[8][8];
float Find_output4[8][8];
float Find_output5[8][8];
void input()
{
    printf("What is going on\n");
    for (int i=0;i<INP_SIZE;i++)
                        {
                            for(int j=0; j<INP_SIZE; j++)
                            {
                                Find_input_A[i][j]= (rand()%20);
                                Find_input_B[i][j]= (rand()%20);
```

```
                            }
                        }
}
void PS()
{
    float Find_outputps[INP_SIZE][INP_SIZE];
    XTime time_PS_start , time_PS_end;
    XTime_SetTime(0);
    XTime_GetTime(&time_PS_start);
    for(int i=0; i<MATSIZE; i++)
    {
        for(int j=0; j<MATSIZE; j++)
        {
            float res=0;
            for(int index=0; index<MATSIZE; index++)
            {
                res+=Find_input_A[i][index]*Find_input_B[index][j];
            }
            Find_outputps[i][j]=res;
        }
    }
    XTime_GetTime(&time_PS_end);
    printf("\n-----------PS FPGA EXECUTION TIME-------------------\n");
                                                    float time_PS = 0;
                                                    time_PS = (float)1.0 *
(time_PS_end - time_PS_start) / (COUNTS_PER_SECOND/1000000);
                                                    printf("Execution Time for
PS in Micro-Seconds : %f\n" , time_PS);
    for(int row=0; row<MATSIZE; row++)
        {
            for( int col=0; col<MATSIZE; col++)
            {
                printf("Input A %f, Input B
%f\n",Find_input_A[row][col],Find_input_B[row][col]);
            }
        }
    for(int row=0; row<MATSIZE; row++)
            {
                for(int col=0; col<MATSIZE; col++)
                {
                    printf("Output %f\n",Find_outputps[row][col]);
                }
            }
}
int Find_ACP1()
{
    float DMA_input[INP_SIZE*INP_SIZE*2];
    float Find_output_DMA[INP_SIZE*INP_SIZE];
```

```
    int status;
     XAxiDma_Config *DMA_confptracp; //DMA configuration pointer
     XAxiDma AxiDMAacp; // DMA instance pointer
     DMA_confptracp = XAxiDma_LookupConfig(XPAR_AXI_DMA_0_DEVICE_ID);
     status = XAxiDma_CfgInitialize(&AxiDMAacp, DMA_confptracp);
     if(status != XST_SUCCESS)
     {
        printf("ACP DMA Init Failed\t\n");
        return XST_FAILURE;
     }
     XTime time_ACP_start , time_ACP_end;
                 XTime_SetTime(0);
                 XTime_GetTime(&time_ACP_start);
            int index=0;
            for(int row=0; row<MATSIZE; row++)
            {
                for(int col=0; col<MATSIZE; col++)
                {
                    DMA_input[index]=Find_input_A[row][col];
                    index++;
                }
            }
            for(int row=0; row<MATSIZE; row++)
            {
                for(int col=0; col<MATSIZE; col++)
                {
                    DMA_input[index]=Find_input_B[row][col];
                    index++;
                }
            }
//          for(int i=0; i<INP_SIZE*INP_SIZE*2; i++)
//          {
//              printf("Input %f \n ",DMA_input[i]);
//          }
               status = XAxiDma_SimpleTransfer(&AxiDMAacp,
(UINTPTR)Find_output_DMA,(sizeof(float)*INP_SIZE*INP_SIZE),XAXIDMA_DEVICE_TO_D
MA);
               status = XAxiDma_SimpleTransfer(&AxiDMAacp, (UINTPTR)DMA_input,
(sizeof(float)*INP_SIZE*INP_SIZE*2),XAXIDMA_DMA_TO_DEVICE);
               status = XAxiDma_ReadReg(XPAR_AXI_DMA_0_BASEADDR,0x04) &
0x00000002;
               while(status!=0x00000002)
               {
                 status = XAxiDma_ReadReg(XPAR_AXI_DMA_0_BASEADDR,0x04) &
0x00000002;
               }
               status = XAxiDma_ReadReg(XPAR_AXI_DMA_0_BASEADDR,0x34) &
0x00000002;
```

```
             while(status!=0x00000002)
             {
               status = XAxiDma_ReadReg(XPAR_AXI_DMA_0_BASEADDR,0x34) &
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);
                                 for(int i = 0 ; i<INP_SIZE*INP_SIZE; i++)
                                 {
                                     printf("Output  :
%f\n",Find_output_DMA[i]);
                                 }
                                 index=0;
                            for(int row=0; row<MATSIZE; row++)
                                       {
                                           for(int col=0; col<MATSIZE; col++)
                                           {
                                               Find_output1[row][col]=Find_ou
tput_DMA[index];
                                               index++;
                                           }
                                       }
     return 0;
}
int Find_ACP2()
{
    float DMA_input[INP_SIZE*INP_SIZE*2];
    float Find_output_DMA[INP_SIZE*INP_SIZE];
    int status;
     XAxiDma_Config *DMA_confptracp; //DMA configuration pointer
     XAxiDma AxiDMAacp; // DMA instance pointer
     DMA_confptracp = XAxiDma_LookupConfig(XPAR_AXI_DMA_1_DEVICE_ID);
     status = XAxiDma_CfgInitialize(&AxiDMAacp, DMA_confptracp);
     if(status != XST_SUCCESS)
     {
        printf("ACP DMA Init Failed\t\n");
        return XST_FAILURE;
     }
     XTime time_ACP_start , time_ACP_end;
                 XTime_SetTime(0);
                 XTime_GetTime(&time_ACP_start);
```

```
            int index=0;
            for(int row=0; row<MATSIZE; row++)
            {
                for(int col=0; col<MATSIZE; col++)
                {
                    DMA_input[index]=Find_input_A[row][col];
                    index++;
                }
            }
            for(int row=0; row<MATSIZE; row++)
            {
                for(int col=0; col<MATSIZE; col++)
                {
                    DMA_input[index]=Find_input_B[row][col];
                    index++;
                }
            }
//          for(int i=0; i<INP_SIZE*INP_SIZE*2; i++)
//          {
//              printf("Input %f \n ",DMA_input[i]);
//          }
               status = XAxiDma_SimpleTransfer(&AxiDMAacp,
(UINTPTR)Find_output_DMA,(sizeof(float)*INP_SIZE*INP_SIZE),XAXIDMA_DEVICE_TO_D
MA);
               status = XAxiDma_SimpleTransfer(&AxiDMAacp, (UINTPTR)DMA_input,
(sizeof(float)*INP_SIZE*INP_SIZE*2),XAXIDMA_DMA_TO_DEVICE);
               status = XAxiDma_ReadReg(XPAR_AXI_DMA_0_BASEADDR,0x04) &
0x00000002;
               while(status!=0x00000002)
               {
                 status = XAxiDma_ReadReg(XPAR_AXI_DMA_0_BASEADDR,0x04) &
0x00000002;
               }
               status = XAxiDma_ReadReg(XPAR_AXI_DMA_0_BASEADDR,0x34) &
0x00000002;

              while(status!=0x00000002)
              {
                status = XAxiDma_ReadReg(XPAR_AXI_DMA_0_BASEADDR,0x34) &
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);
                                for(int i = 0 ; i<INP_SIZE*INP_SIZE; i++)
                                {
                                        printf("Output  :
%f\n",Find_output_DMA[i]);
                                }
                                index=0;
                            for(int row=0; row<MATSIZE; row++)
                                    {
                                       for(int col=0; col<MATSIZE; col++)
                                       {
                                           Find_output2[row][col]=Find_ou
tput_DMA[index];

                                           index++;
                                       }
                                    }
     return 0;
}
int Find_ACP3()
{
    float DMA_input[INP_SIZE*INP_SIZE*2];
    float Find_output_DMA[INP_SIZE*INP_SIZE];
    int status;
     XAxiDma_Config *DMA_confptracp; //DMA configuration pointer
     XAxiDma AxiDMAacp; // DMA instance pointer
     DMA_confptracp = XAxiDma_LookupConfig(XPAR_AXI_DMA_2_DEVICE_ID);
     status = XAxiDma_CfgInitialize(&AxiDMAacp, DMA_confptracp);
     if(status != XST_SUCCESS)
     {
        printf("ACP DMA Init Failed\t\n");
        return XST_FAILURE;
     }
     XTime time_ACP_start , time_ACP_end;
                 XTime_SetTime(0);
                 XTime_GetTime(&time_ACP_start);
            int index=0;
            for(int row=0; row<MATSIZE; row++)
            {
                for(int col=0; col<MATSIZE; col++)
                {
                    DMA_input[index]=Find_input_A[row][col];
                    index++;
                }
            }
            for(int row=0; row<MATSIZE; row++)
            {
                for(int col=0; col<MATSIZE; col++)
```

```
                {
                    DMA_input[index]=Find_input_B[row][col];
                    index++;
                }
            }
//          for(int i=0; i<INP_SIZE*INP_SIZE*2; i++)
//          {
//              printf("Input %f \n ",DMA_input[i]);
//          }
               status = XAxiDma_SimpleTransfer(&AxiDMAacp,
(UINTPTR)Find_output_DMA,(sizeof(float)*INP_SIZE*INP_SIZE),XAXIDMA_DEVICE_TO_D
MA);
               status = XAxiDma_SimpleTransfer(&AxiDMAacp, (UINTPTR)DMA_input,
(sizeof(float)*INP_SIZE*INP_SIZE*2),XAXIDMA_DMA_TO_DEVICE);
               status = XAxiDma_ReadReg(XPAR_AXI_DMA_0_BASEADDR,0x04) &
0x00000002;
               while(status!=0x00000002)
               {
                 status = XAxiDma_ReadReg(XPAR_AXI_DMA_0_BASEADDR,0x04) &
0x00000002;
               }
               status = XAxiDma_ReadReg(XPAR_AXI_DMA_0_BASEADDR,0x34) &
0x00000002;

              while(status!=0x00000002)
              {
                status = XAxiDma_ReadReg(XPAR_AXI_DMA_0_BASEADDR,0x34) &
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);
                                for(int i = 0 ; i<INP_SIZE*INP_SIZE; i++)
                                {
                                        printf("Output  :
%f\n",Find_output_DMA[i]);
                                }
                                index=0;
                           for(int row=0; row<MATSIZE; row++)
                                    {
                                       for(int col=0; col<MATSIZE; col++)
                                       {
```

```
                                                Find_output3[row][col]=Find_ou
tput_DMA[index];
                                                index++;
                                            }
                                        }
     return 0;
}
int Find_ACP4()
{
    float DMA_input[INP_SIZE*INP_SIZE*2];
    float Find_output_DMA[INP_SIZE*INP_SIZE];
    int status;
     XAxiDma_Config *DMA_confptracp; //DMA configuration pointer
     XAxiDma AxiDMAacp; // DMA instance pointer
     DMA_confptracp = XAxiDma_LookupConfig(XPAR_AXI_DMA_3_DEVICE_ID);
     status = XAxiDma_CfgInitialize(&AxiDMAacp, DMA_confptracp);
     if(status != XST_SUCCESS)
     {
        printf("ACP DMA Init Failed\t\n");
        return XST_FAILURE;
     }
     XTime time_ACP_start , time_ACP_end;
                 XTime_SetTime(0);
                 XTime_GetTime(&time_ACP_start);
            int index=0;
            for(int row=0; row<MATSIZE; row++)
            {
                for(int col=0; col<MATSIZE; col++)
                {
                    DMA_input[index]=Find_input_A[row][col];
                    index++;
                }
            }
            for(int row=0; row<MATSIZE; row++)
            {
                for(int col=0; col<MATSIZE; col++)
                {
                    DMA_input[index]=Find_input_B[row][col];
                    index++;
                }
            }
//          for(int i=0; i<INP_SIZE*INP_SIZE*2; i++)
//          {
//              printf("Input %f \n ",DMA_input[i]);
//          }
               status = XAxiDma_SimpleTransfer(&AxiDMAacp,
(UINTPTR)Find_output_DMA,(sizeof(float)*INP_SIZE*INP_SIZE),XAXIDMA_DEVICE_TO_D
MA);
```

```
               status = XAxiDma_SimpleTransfer(&AxiDMAacp, (UINTPTR)DMA_input,
(sizeof(float)*INP_SIZE*INP_SIZE*2),XAXIDMA_DMA_TO_DEVICE);
               status = XAxiDma_ReadReg(XPAR_AXI_DMA_0_BASEADDR,0x04) &
0x00000002;
               while(status!=0x00000002)
               {
                 status = XAxiDma_ReadReg(XPAR_AXI_DMA_0_BASEADDR,0x04) &
0x00000002;
               }
               status = XAxiDma_ReadReg(XPAR_AXI_DMA_0_BASEADDR,0x34) &
0x00000002;

              while(status!=0x00000002)
              {
                status = XAxiDma_ReadReg(XPAR_AXI_DMA_0_BASEADDR,0x34) &
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);
                                for(int i = 0 ; i<INP_SIZE*INP_SIZE; i++)
                                {
                                        printf("Output  :
%f\n",Find_output_DMA[i]);
                                }
                                index=0;
                           for(int row=0; row<MATSIZE; row++)
                                    {
                                       for(int col=0; col<MATSIZE; col++)
                                       {
                                           Find_output4[row][col]=Find_ou
tput_DMA[index];
                                           index++;
                                       }
                                    }
     return 0;
}
int Find_ACP5()
{
    float DMA_input[INP_SIZE*INP_SIZE*2];
    float Find_output_DMA[INP_SIZE*INP_SIZE];
    int status;
     XAxiDma_Config *DMA_confptracp; //DMA configuration pointer
```

```
    XAxiDma AxiDMAacp; // DMA instance pointer
    DMA_confptracp = XAxiDma_LookupConfig(XPAR_AXI_DMA_4_DEVICE_ID);
    status = XAxiDma_CfgInitialize(&AxiDMAacp, DMA_confptracp);
    if(status != XST_SUCCESS)
    {
       printf("ACP DMA Init Failed\t\n");
       return XST_FAILURE;
    }
    XTime time_ACP_start , time_ACP_end;
                XTime_SetTime(0);
                XTime_GetTime(&time_ACP_start);
            int index=0;
            for(int row=0; row<MATSIZE; row++)
            {
                for(int col=0; col<MATSIZE; col++)
                {
                    DMA_input[index]=Find_input_A[row][col];
                    index++;
                }
            }
            for(int row=0; row<MATSIZE; row++)
            {
                for(int col=0; col<MATSIZE; col++)
                {
                    DMA_input[index]=Find_input_B[row][col];
                    index++;
                }
            }
//          for(int i=0; i<INP_SIZE*INP_SIZE*2; i++)
//          {
//              printf("Input %f \n ",DMA_input[i]);
//          }
               status = XAxiDma_SimpleTransfer(&AxiDMAacp,
(UINTPTR)Find_output_DMA,(sizeof(float)*INP_SIZE*INP_SIZE),XAXIDMA_DEVICE_TO_D
MA);
               status = XAxiDma_SimpleTransfer(&AxiDMAacp, (UINTPTR)DMA_input,
(sizeof(float)*INP_SIZE*INP_SIZE*2),XAXIDMA_DMA_TO_DEVICE);
               status = XAxiDma_ReadReg(XPAR_AXI_DMA_0_BASEADDR,0x04) &
0x00000002;
               while(status!=0x00000002)
               {
                 status = XAxiDma_ReadReg(XPAR_AXI_DMA_0_BASEADDR,0x04) &
0x00000002;
               }
               status = XAxiDma_ReadReg(XPAR_AXI_DMA_0_BASEADDR,0x34) &
0x00000002;

              while(status!=0x00000002)
```

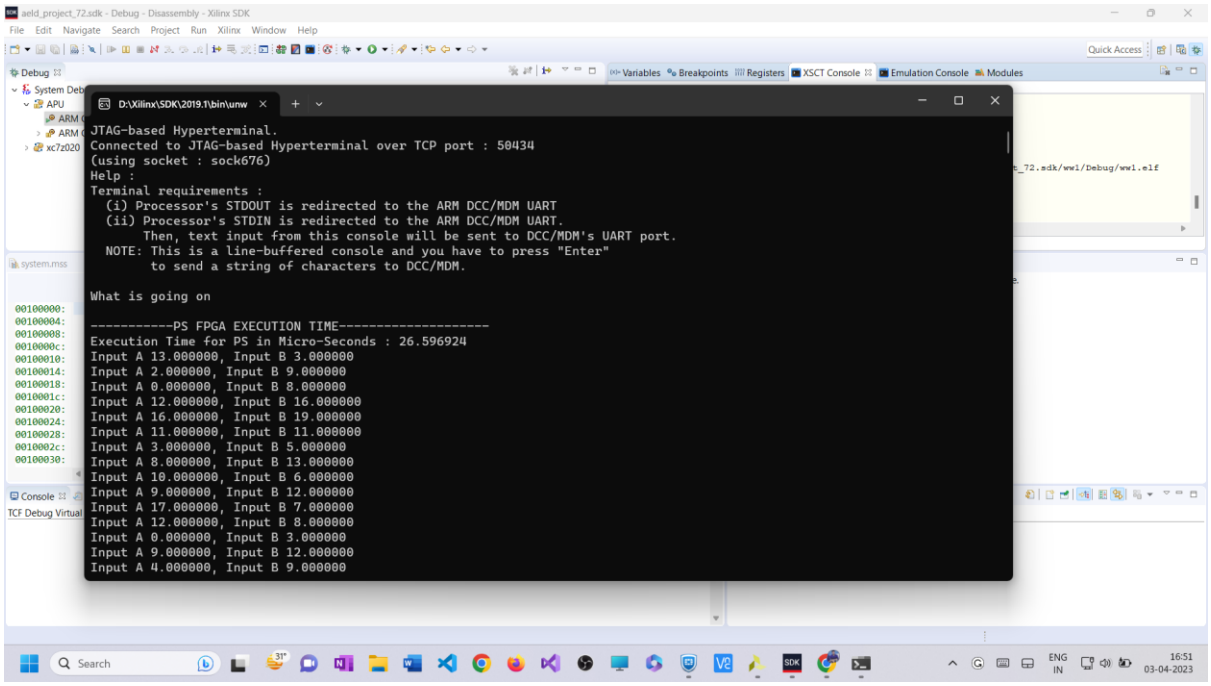
```
              {
                status = XAxiDma_ReadReg(XPAR_AXI_DMA_0_BASEADDR,0x34) &
0x00000002;
              }
              XTime_GetTime(&time_ACP_end);
                            printf("\n-----------ACP FPGA EXECUTION TIME----
----------------\n");
                                float 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);
                                for(int i = 0 ; i<INP_SIZE*INP_SIZE; i++)
                                {
                                        printf("Output  :
%f\n",Find_output_DMA[i]);
                                }
                                index=0;
                           for(int row=0; row<MATSIZE; row++)
                                    {
                                       for(int col=0; col<MATSIZE; col++)
                                       {
                                           Find_output5[row][col]=Find_ou
tput_DMA[index];

                                           index++;
                                       }
                                    }
     return 0;
}
int compare1()
{
    for(int i=0; i<MATSIZE; i++)
    {
        for(int j=0; j<MATSIZE; j++)
        {
            if(Find_output1[i][j]!=Find_outputps[i][j])
            {
                return 0;
            }
        }
    }
    return 1;
}
int compare2()
{
    for(int i=0; i<MATSIZE; i++)
    {
        for(int j=0; j<MATSIZE; j++)
```

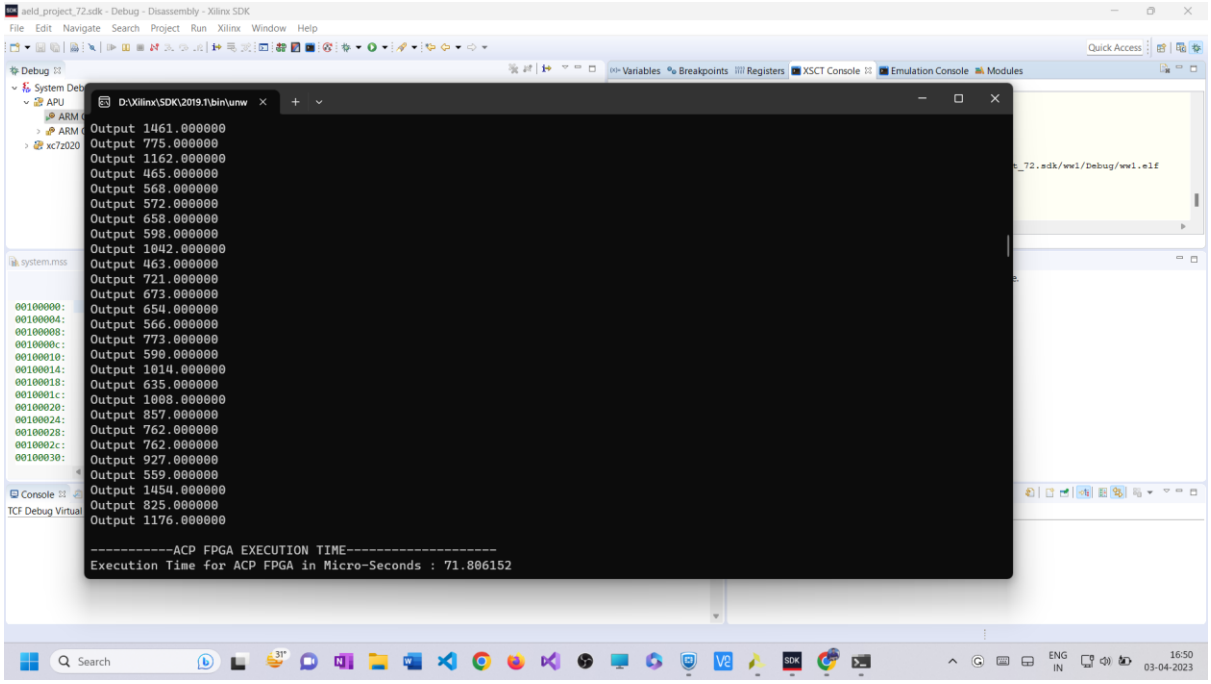
```
        {
            if(Find_output2[i][j]!=Find_outputps[i][j])
            {
                return 0;
            }
        }
    }
    return 1;
}
int compare3()
{
    for(int i=0; i<MATSIZE; i++)
    {
        for(int j=0; j<MATSIZE; j++)
        {
            if(Find_output3[i][j]!=Find_outputps[i][j])
            {
                return 0;
            }
        }
    }
    return 1;
}
int compare4()
{
    for(int i=0; i<MATSIZE; i++)
    {
        for(int j=0; j<MATSIZE; j++)
        {
            if(Find_output4[i][j]!=Find_outputps[i][j])
            {
                return 0;
            }
        }
    }
    return 1;
}
int compare5()
{
    for(int i=0; i<MATSIZE; i++)
    {
        for(int j=0; j<MATSIZE; j++)
        {
            if(Find_output5[i][j]!=Find_outputps[i][j])
            {
                return 0;
            }
        }
```

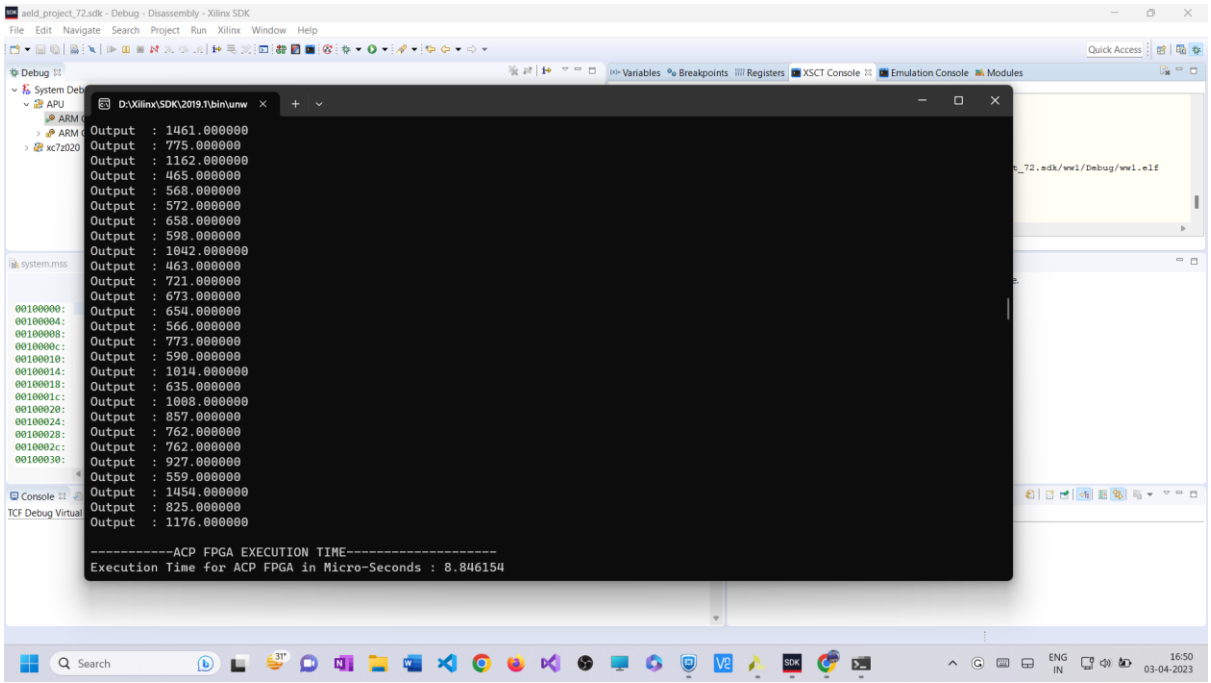
```
    }
    return 1;
}
int main()
{
    init_platform();
    input();
    PS();
    Find_ACP1();
    Find_ACP2();
    Find_ACP3();
    Find_ACP4();
    Find_ACP5();
    int ans=compare1();
    int ans1=compare2();
    int ans2=compare3();
    int ans3=compare4();
    int ans4=compare5();
    if(ans==1)
    {
        printf("You are great\n");
    }
    else
    {
        printf("No good at all\n");
    }
    if(ans1==1)
    {
        printf("You are great\n");
    }
    else
    {
        printf("No good at all");
    }
    if(ans2==1)
    {
        printf("You are great\n");
    }
    else
    {
        printf("No good at all\n");
    }
    if(ans3==1)
    {
        printf("You are great\n");
    }
    else
    {
```

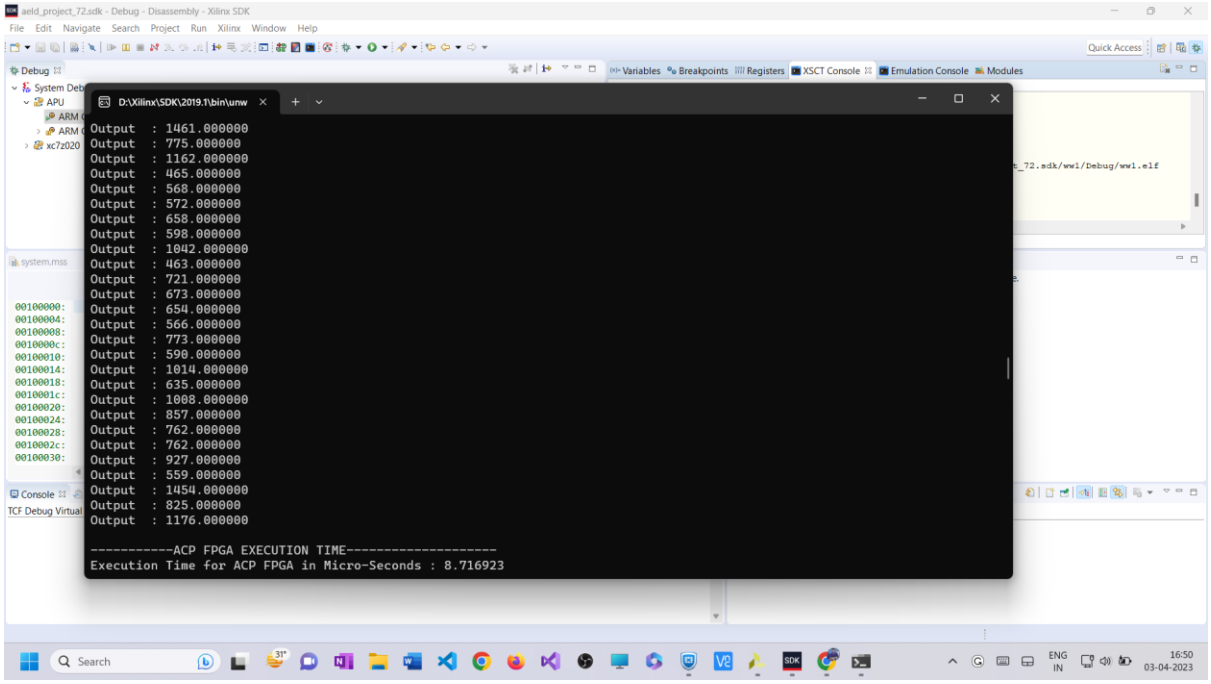
```
        printf("No good at all\n");
    }
    if(ans4==1)
    {
        printf("You are great\n");
    }
    else
    {
        printf("No good at all");
    }
    cleanup_platform();
    return 0;
}
```

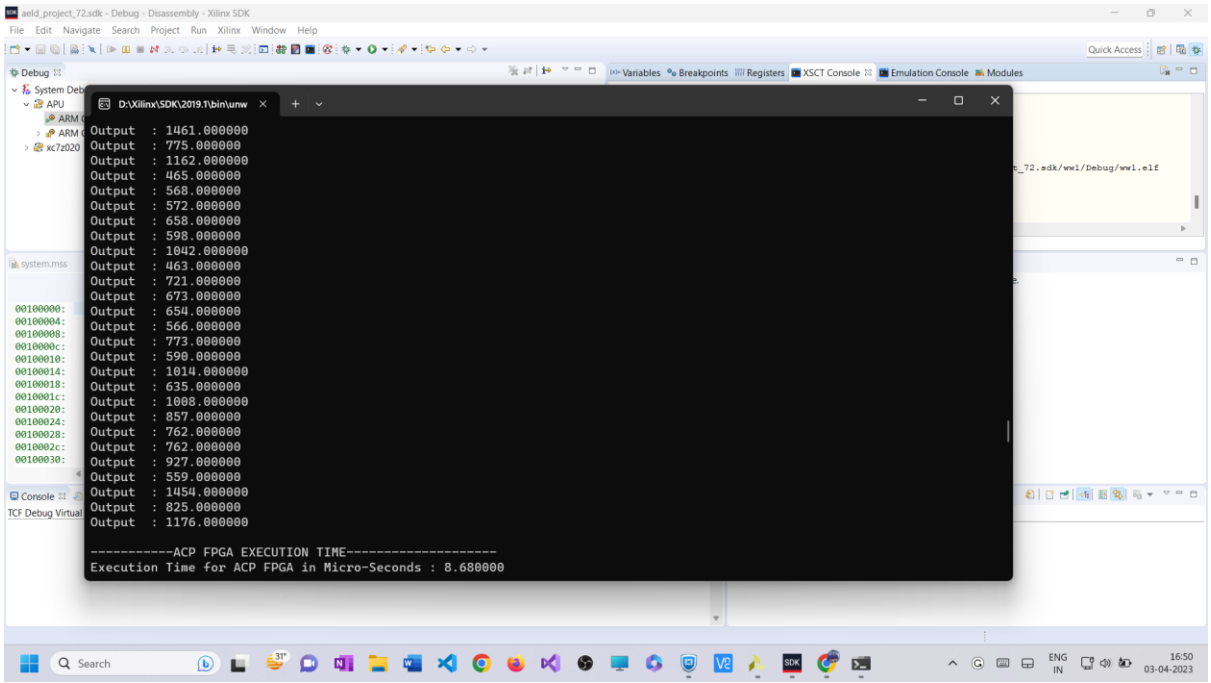
Jtagterminal:-

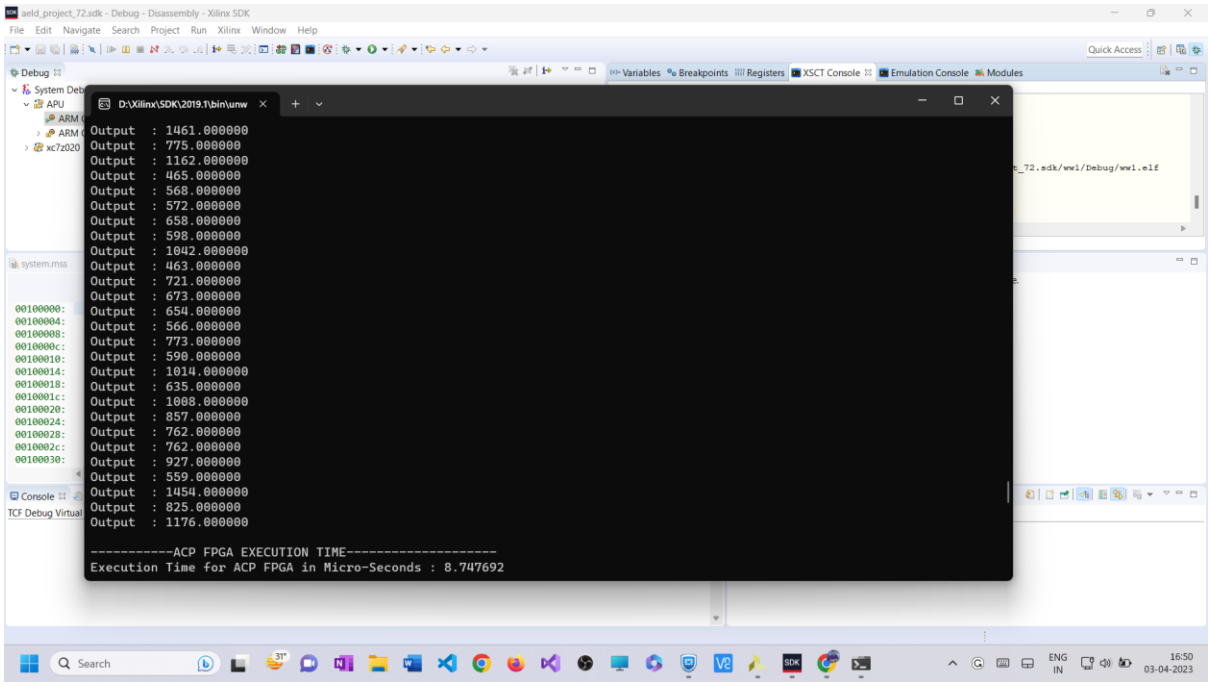



D:\Xilinx\SDK\2019.1\bin\unw
Output 1461.000000
Output 775.000000
Output 1162.000000
Output 465.000000
Output 568.000000
Output 572.000000
Output 658.000000
Output 598.000000
Output 1042.000000
Output 463.000000
Output 721.000000
Output 673.000000
Output 654.000000
Output 566.000000
Output 773.000000
Output 590.000000
Output 1014.000000
Output 635.000000
Output 1008.000000
Output 857.000000
Output 762.000000
Output 762.000000
Output 927.000000
Output 559.000000
Output 1454.000000
Output 825.000000
Output 1176.000000
----------ACP FPGA EXECUTION TIME----------
Execution Time for ACP FPGA in Micro-Seconds : 71.806152

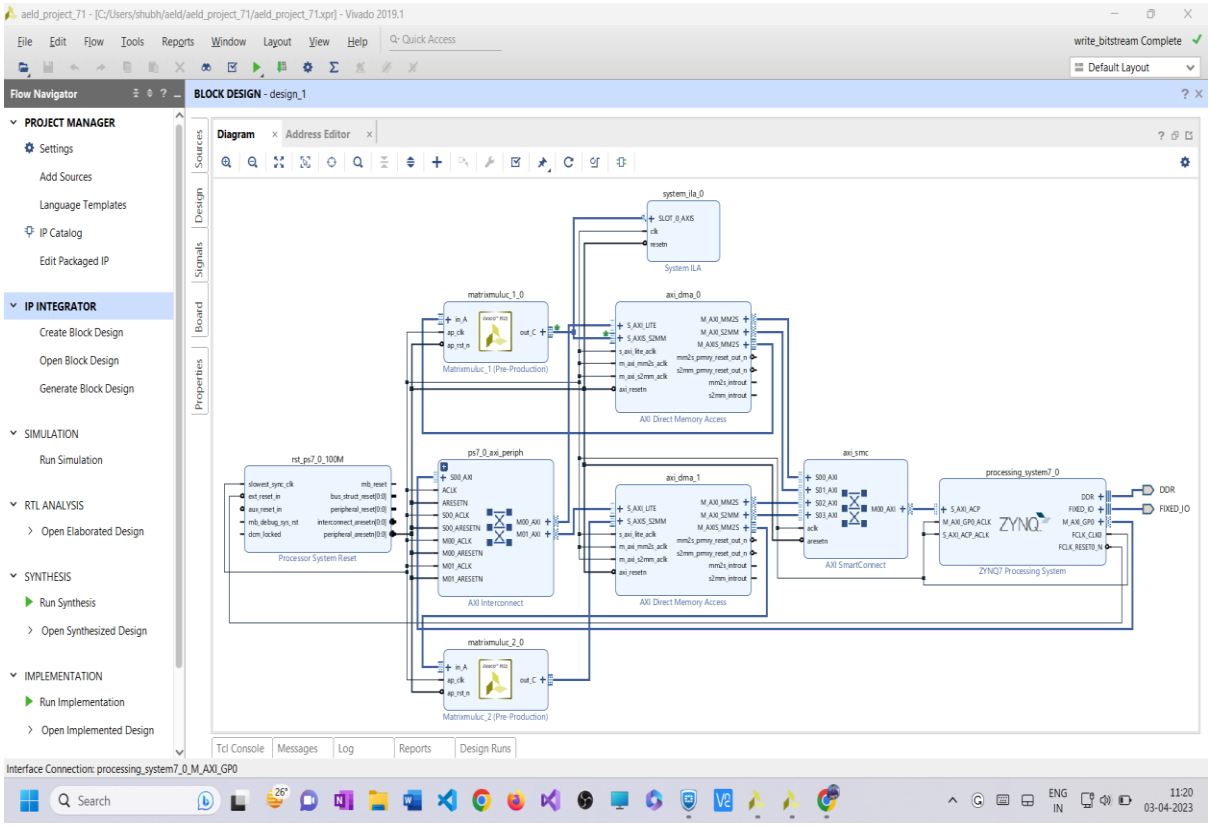

D:\Xilinx\SDK\2019.1\bin\unw
Output : 1461.000000
Output : 775.000000
Output : 1162.000000
Output : 465.000000
Output : 568.000000
Output : 572.000000
Output : 658.000000
Output : 598.000000
Output : 1042.000000
Output : 463.000000
Output : 721.000000
Output : 673.000000
Output : 654.000000
Output : 566.000000
Output : 773.000000
Output : 590.000000
Output : 1014.000000
Output : 635.000000
Output : 1008.000000
Output : 857.000000
Output : 762.000000
Output : 762.000000
Output : 927.000000
Output : 559.000000
Output : 1454.000000
Output : 825.000000
Output : 1176.000000
----------ACP FPGA EXECUTION TIME----------
Execution Time for ACP FPGA in Micro-Seconds : 8.846154


D:\Xilinx\SDK\2019.1\bin\unw
Output : 1461.000000
Output : 775.000000
Output : 1162.000000
Output : 465.000000
Output : 568.000000
Output : 572.000000
Output : 658.000000
Output : 598.000000
Output : 1042.000000
Output : 463.000000
Output : 721.000000
Output : 673.000000
Output : 654.000000
Output : 566.000000
Output : 773.000000
Output : 590.000000
Output : 1014.000000
Output : 635.000000
Output : 1008.000000
Output : 857.000000
Output : 762.000000
Output : 762.000000
Output : 927.000000
Output : 559.000000
Output : 1454.000000
Output : 825.000000
Output : 1176.000000
----------ACP FPGA EXECUTION TIME----------
Execution Time for ACP FPGA in Micro-Seconds : 8.716923


D:\Xilinx\SDK\2019.1\bin\unw
Output : 1461.000000
Output : 775.000000
Output : 1162.000000
Output : 465.000000
Output : 568.000000
Output : 572.000000
Output : 658.000000
Output : 598.000000
Output : 1042.000000
Output : 463.000000
Output : 721.000000
Output : 673.000000
Output : 654.000000
Output : 566.000000
Output : 773.000000
Output : 590.000000
Output : 1014.000000
Output : 635.000000
Output : 1008.000000
Output : 857.000000
Output : 762.000000
Output : 762.000000
Output : 927.000000
Output : 559.000000
Output : 1454.000000
Output : 825.000000
Output : 1176.000000
----------ACP FPGA EXECUTION TIME----------
Execution Time for ACP FPGA in Micro-Seconds : 8.680000



Block Diagram:-



Code:-

```
/******************************************************************************
*
*
* Copyright (C) 2009 - 2014 Xilinx, Inc.  All rights reserved.
*
* Permission is hereby granted, free of charge, to any person obtaining a copy
* of this software and associated documentation files (the "Software"), to
deal
* in the Software without restriction, including without limitation the rights
* to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
* copies of the Software, and to permit persons to whom the Software is
* furnished to do so, subject to the following conditions:
*
* The above copyright notice and this permission notice shall be included in
* all copies or substantial portions of the Software.
*
* Use of the Software is limited solely to applications:
* (a) running on a Xilinx device, or
* (b) that interact with a Xilinx device through a bus or interconnect.
*
* THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
* IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
* FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL
* XILINX  BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY,
* WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF
* OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
* SOFTWARE.
*
* Except as contained in this notice, the name of the Xilinx shall not be used
* in advertising or otherwise to promote the sale, use or other dealings in
* this Software without prior written authorization from Xilinx.
*
******************************************************************************
/

/*
 * helloworld.c: simple test application
 *
 * This application configures UART 16550 to baud rate 9600.
 * PS7 UART (Zynq) is not initialized by this application, since
 * bootrom/bsp configures it to baud rate 115200
 *
 * ------------------------------------------------
 * | UART TYPE   BAUD RATE                        |
 * ------------------------------------------------
```

```
 *   uartns550   9600
 *   uartlite    Configurable only in HW design
 *   ps7_uart    115200 (configured by bootrom/bsp)
 */

#include <stdio.h>
#include <stdlib.h>
#include "xaxidma.h"
#include "xparameters.h"
#include "platform.h"
#include<xtime_l.h>
#include<complex.h>
#define INP_SIZE 32
#define MATSIZE 32
float complex Find_input_A[32][32];
float complex Find_input_B[32][32];
float complex Find_outputps[32][32];
float complex Find_output1[32][32];
float complex Find_output2[32][32];
void input()
{
    for (int i=0;i<INP_SIZE;i++)
                        {
                            for(int j=0; j<INP_SIZE; j++)
                            {
                                Find_input_A[i][j]= (rand()%20)+I*(rand()%20);
                                Find_input_B[i][j]= (rand()%20)+I*(rand()%20);
                            }
                        }
}
void PS()
{
    float complex Find_outputps[INP_SIZE][INP_SIZE];
    XTime time_PS_start , time_PS_end;
    XTime_SetTime(0);
    XTime_GetTime(&time_PS_start);
    for(int i=0; i<MATSIZE; i++)
    {
        for(int j=0; j<MATSIZE; j++)
        {
            float complex res=0;
            for(int index=0; index<MATSIZE; index++)
            {
                res+=Find_input_A[i][index]*Find_input_B[index][j];
            }
            Find_outputps[i][j]=res;
        }
    }
```

```
    XTime_GetTime(&time_PS_end);
    for(int row=0; row<MATSIZE; row++)
        {
            for( int col=0; col<MATSIZE; col++)
            {
                printf("Input A %lf %lf, Input B %lf
%lf\n",creal(Find_input_A[row][col]),cimag(Find_input_A[row][col]),creal(Find_
input_B[row][col]),cimag(Find_input_B[row][col]));
            }
        }
    for(int row=0; row<MATSIZE; row++)
            {
                for(int col=0; col<MATSIZE; col++)
                {
                    printf("Output %lf
%lf\n",creal(Find_outputps[row][col]),cimag(Find_outputps[row][col]));
                }
            }
    printf("\n----------PS FPGA EXECUTION TIME-------------------\n");
                                        float time_PS = 0;
                                        time_PS = (float)1.0 * (time_PS_end
- time_PS_start) / (COUNTS_PER_SECOND/1000000);
                                        printf("Execution Time for PS in
Micro-Seconds : %f\n" , time_PS);

}
int Find_ACP1()
{
    printf("on first one\n");
    float complex DMA_input[INP_SIZE*INP_SIZE*2];
    float complex Find_output_DMA[INP_SIZE*INP_SIZE];
    int status;
     XAxiDma_Config *DMA_confptracp; //DMA configuration pointer
     XAxiDma AxiDMAacp; // DMA instance pointer
     DMA_confptracp = XAxiDma_LookupConfig(XPAR_AXI_DMA_0_DEVICE_ID);
     status = XAxiDma_CfgInitialize(&AxiDMAacp, DMA_confptracp);
     if(status != XST_SUCCESS)
     {
        printf("ACP DMA Init Failed\t\n");
        return XST_FAILURE;
     }
     XTime time_ACP_start , time_ACP_end;
                 XTime_SetTime(0);
                 XTime_GetTime(&time_ACP_start);
            int index=0;
            for(int row=0; row<MATSIZE; row++)
            {
                for(int col=0; col<MATSIZE; col++)
```

```
                {
                    DMA_input[index]=Find_input_A[row][col];
                    index++;
                }
            }
            for(int row=0; row<MATSIZE; row++)
            {
                for(int col=0; col<MATSIZE; col++)
                {
                    DMA_input[index]=Find_input_B[row][col];
                    index++;
                }
            }
//          for(int i=0; i<INP_SIZE*INP_SIZE*2; i++)
//          {
//              printf("Input %f \n ",DMA_input[i]);
//          }
               status = XAxiDma_SimpleTransfer(&AxiDMAacp,
(UINTPTR)Find_output_DMA,(sizeof(float)*2*INP_SIZE*INP_SIZE),XAXIDMA_DEVICE_TO
_DMA);
               status = XAxiDma_SimpleTransfer(&AxiDMAacp, (UINTPTR)DMA_input,
(sizeof(float)*2*INP_SIZE*INP_SIZE*2),XAXIDMA_DMA_TO_DEVICE);
               status = XAxiDma_ReadReg(XPAR_AXI_DMA_0_BASEADDR,0x04) &
0x00000002;
               while(status!=0x00000002)
               {
                 status = XAxiDma_ReadReg(XPAR_AXI_DMA_0_BASEADDR,0x04) &
0x00000002;
               }
               status = XAxiDma_ReadReg(XPAR_AXI_DMA_0_BASEADDR,0x34) &
0x00000002;

              while(status!=0x00000002)
              {
                status = XAxiDma_ReadReg(XPAR_AXI_DMA_0_BASEADDR,0x34) &
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);
//                                  for(int i = 0 ; i<INP_SIZE*INP_SIZE; i++)
//                                  {
```

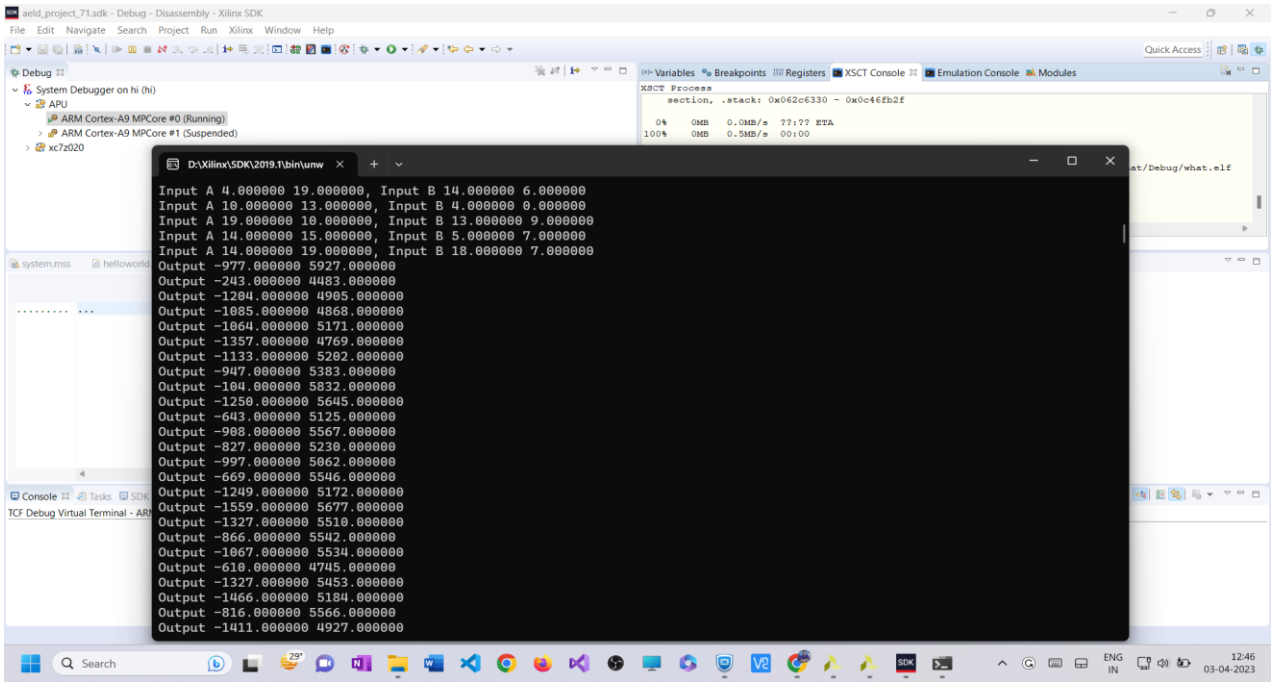
```
//                                    printf("Output   : %lf
%lf\n",creal(Find_output_DMA[i]),cimag(Find_output_DMA[i]));
//                                }
                            printf("H4\n");
                                  index=0;
                            for(int row=0; row<MATSIZE; row++)
                                        {
                                            for(int col=0; col<MATSIZE; col++)
                                            {
                                                Find_output1[row][col]=Find_ou
tput_DMA[index];

                                                index++;
                                            }
                                        }
     return 0;
}
int Find_ACP2()
{
    printf("on second one\n");
    float complex DMA_input[INP_SIZE*INP_SIZE*2];
    float complex Find_output_DMA[INP_SIZE*INP_SIZE];
    int status;
     XAxiDma_Config *DMA_confptracp; //DMA configuration pointer
     XAxiDma AxiDMAacp; // DMA instance pointer
     DMA_confptracp = XAxiDma_LookupConfig(XPAR_AXI_DMA_1_DEVICE_ID);
     status = XAxiDma_CfgInitialize(&AxiDMAacp, DMA_confptracp);
     if(status != XST_SUCCESS)
     {
        printf("ACP DMA Init Failed\t\n");
        return XST_FAILURE;
     }
     XTime time_ACP_start , time_ACP_end;
                 XTime_SetTime(0);
                 XTime_GetTime(&time_ACP_start);
            int index=0;
            for(int row=0; row<MATSIZE; row++)
            {
                for(int col=0; col<MATSIZE; col++)
                {
                    DMA_input[index]=Find_input_A[row][col];
                    index++;
                }
            }
            for(int row=0; row<MATSIZE; row++)
            {
                for(int col=0; col<MATSIZE; col++)
                {
                    DMA_input[index]=Find_input_B[row][col];
```

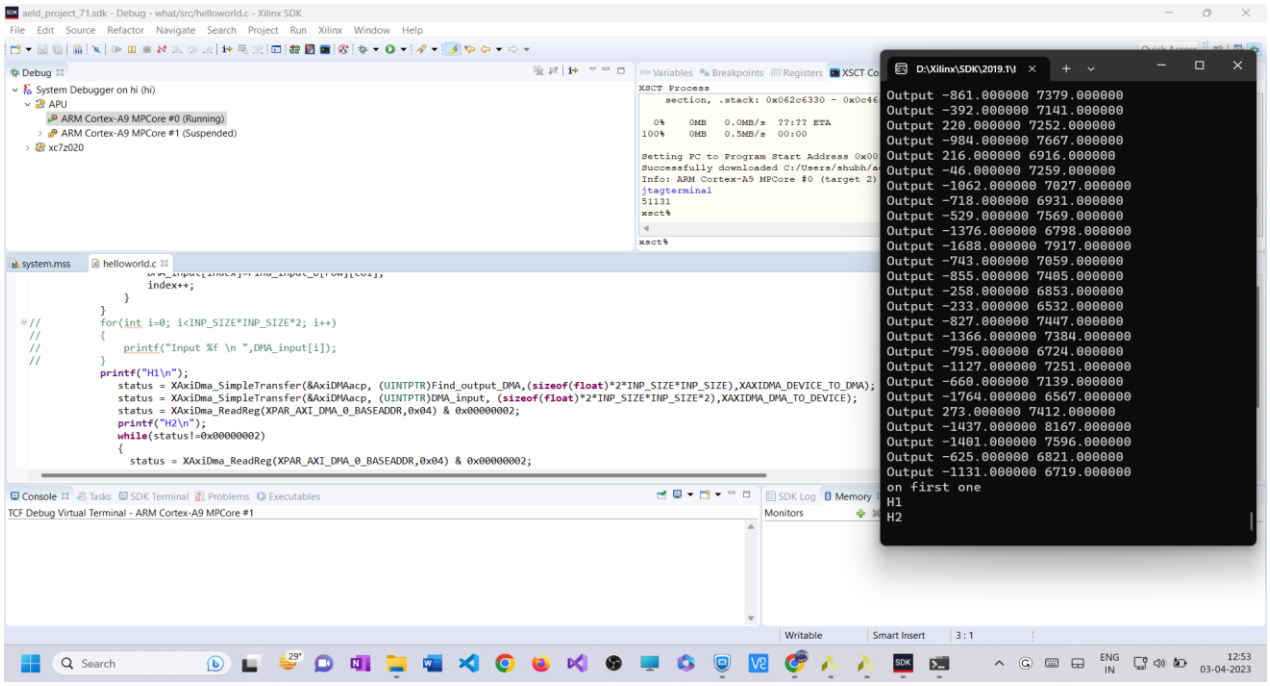
```
                    index++;
                }
            }
//          for(int i=0; i<INP_SIZE*INP_SIZE*2; i++)
//          {
//              printf("Input %f \n ",DMA_input[i]);
//          }
               status = XAxiDma_SimpleTransfer(&AxiDMAacp,
(UINTPTR)Find_output_DMA,(sizeof(float)*2*INP_SIZE*INP_SIZE),XAXIDMA_DEVICE_TO
_DMA);
               status = XAxiDma_SimpleTransfer(&AxiDMAacp, (UINTPTR)DMA_input,
(sizeof(float)*2*INP_SIZE*INP_SIZE*2),XAXIDMA_DMA_TO_DEVICE);
               status = XAxiDma_ReadReg(XPAR_AXI_DMA_0_BASEADDR,0x04) &
0x00000002;
               while(status!=0x00000002)
               {
                 status = XAxiDma_ReadReg(XPAR_AXI_DMA_0_BASEADDR,0x04) &
0x00000002;
               }
               status = XAxiDma_ReadReg(XPAR_AXI_DMA_0_BASEADDR,0x34) &
0x00000002;

              while(status!=0x00000002)
              {
                status = XAxiDma_ReadReg(XPAR_AXI_DMA_0_BASEADDR,0x34) &
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);
//                                  for(int i = 0 ; i<INP_SIZE*INP_SIZE; i++)
//                                  {
//                                          printf("Output   : %lf
%lf\n",creal(Find_output_DMA[i]),cimag(Find_output_DMA[i]));
//                                  }
                                    index=0;
                               for(int row=0; row<MATSIZE; row++)
                                        {
                                            for(int col=0; col<MATSIZE; col++)
                                            {
                                                Find_output2[row][col]=Find_ou
tput_DMA[index];
                                                index++;
```

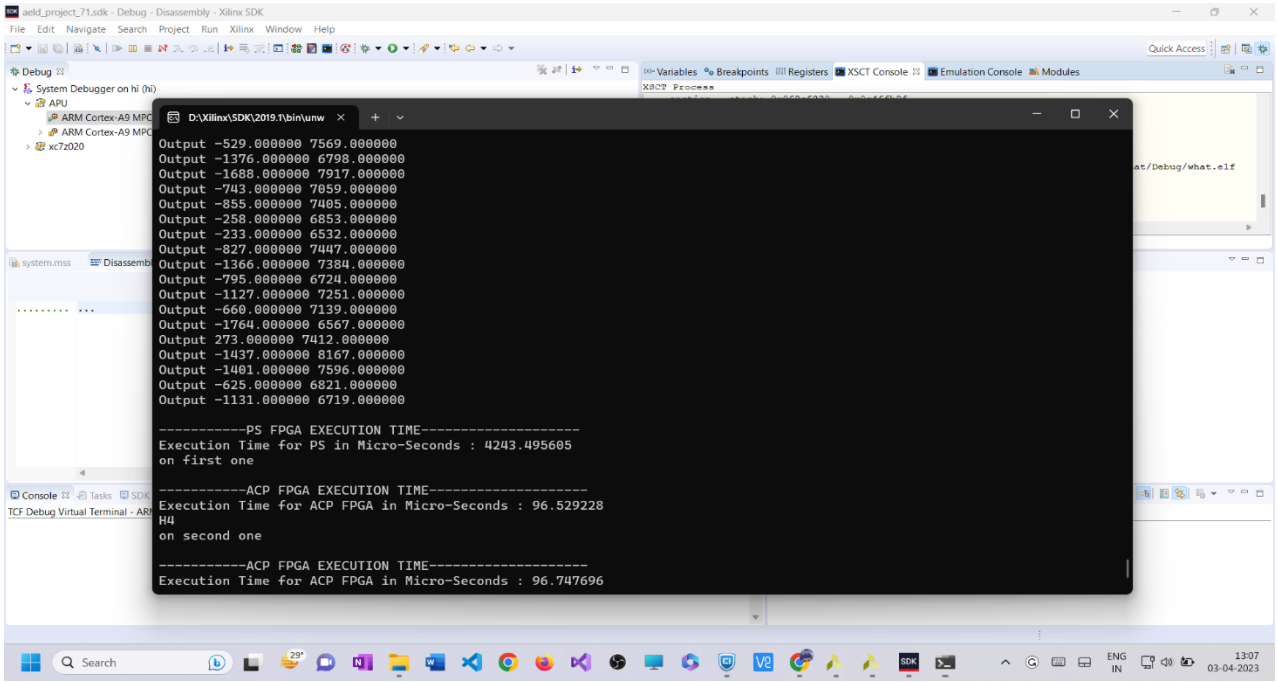
```
                                            }
                                        }
     return 0;
}
int compare1()
{
    for(int i=0; i<MATSIZE; i++)
    {
        for(int j=0; j<MATSIZE; j++)
        {
            if(Find_output1[i][j]!=Find_outputps[i][j])
            {
                return 0;
            }
        }
    }
    return 1;
}
int compare2()
{
    for(int i=0; i<MATSIZE; i++)
    {
        for(int j=0; j<MATSIZE; j++)
        {
            if(Find_output2[i][j]!=Find_outputps[i][j])
            {
                return 0;
            }
        }
    }
    return 1;
}
int main()
{
    init_platform();
    input();
    PS();
    Find_ACP1();
    Find_ACP2();
    int ans=compare1();
    int ans1=compare2();
    if(ans==1)
    {
        printf("You are great\n");
    }
    else
    {
        printf("No good at all\n");
```

```
    }
    if(ans1==1)
        {
            printf("You are great\n");
        }
        else
        {
            printf("No good at all");
        }
    cleanup_platform();
    return 0;
}
```

Jtagterminal:-




aeld_project_71.sdk - Debug - what/src/helloworld.c - Xilinx SDK
File Edit Source Refactor Navigate Search Project Run Xilinx Window Help
Debug
System Debugger on hi (hi)
APU
ARM Cortex-A9 MPCore #0 (Running)
ARM Cortex-A9 MPCore #1 (Suspended)
xc7z020
D:\Xilinx\SDK\2019.1\i
Output -861.000000 7379.000000
Output -392.000000 7141.000000
Output 220.000000 7252.000000
Output -984.000000 7667.000000
Output 216.000000 6916.000000
Output -46.000000 7259.000000
Output -1062.000000 7027.000000
Output -718.000000 6931.000000
Output -529.000000 7569.000000
Output -1376.000000 6798.000000
Output -1688.000000 7917.000000
Output -743.000000 7059.000000
Output -855.000000 7405.000000
Output -258.000000 6853.000000
Output -233.000000 6532.000000
Output -827.000000 7447.000000
Output -1366.000000 7384.000000
Output -795.000000 6724.000000
Output -1127.000000 7251.000000
Output -660.000000 7139.000000
Output -1764.000000 6567.000000
Output 273.000000 7412.000000
Output -1437.000000 8167.000000
Output -1401.000000 7596.000000
Output -625.000000 6821.000000
Output -1131.000000 6719.000000
on first one
H1
H2
system.mss
helloworld.c
printf("H1\n");
status = XAxiDma_SimpleTransfer(&AxiDMAacp, (UINTPTR)Find_output_DMA,(sizeof(float)*2*INP_SIZE*INP_SIZE),XAXIDMA_DEVICE_TO_DMA);
status = XAxiDma_SimpleTransfer(&AxiDMAacp, (UINTPTR)DMA_input, (sizeof(float)*2*INP_SIZE*INP_SIZE*2),XAXIDMA_DMA_TO_DEVICE);
status = XAxiDma_ReadReg(XPAR_AXI_DMA_0_BASEADDR,0x04) & 0x00000002;
printf("H2\n");
while(status!=0x00000002)
status = XAxiDma_ReadReg(XPAR_AXI_DMA_0_BASEADDR,0x04) & 0x00000002;
Console
TCF Debug Virtual Terminal - ARM Cortex-A9 MPCore #1
SDK Log
Memory
Monitors
Writable
Smart Insert
3 : 1
Search
ENG IN
12:53
03-04-2023


aeld_project_71.sdk - Debug - Disassembly - Xilinx SDK
File Edit Navigate Search Project Run Xilinx Window Help
Debug
System Debugger on hi (hi)
APU
xc7z020
D:\Xilinx\SDK\2019.1\bin\unw
Output -529.000000 7569.000000
Output -1376.000000 6798.000000
Output -1688.000000 7917.000000
Output -743.000000 7059.000000
Output -855.000000 7405.000000
Output -258.000000 6853.000000
Output -233.000000 6532.000000
Output -827.000000 7447.000000
Output -1366.000000 7384.000000
Output -795.000000 6724.000000
Output -1127.000000 7251.000000
Output -660.000000 7139.000000
Output -1764.000000 6567.000000
Output 273.000000 7412.000000
Output -1437.000000 8167.000000
Output -1401.000000 7596.000000
Output -625.000000 6821.000000
Output -1131.000000 6719.000000
----------PS FPGA EXECUTION TIME--------------------
Execution Time for PS in Micro-Seconds : 4243.495605
on first one
----------ACP FPGA EXECUTION TIME--------------------
Execution Time for ACP FPGA in Micro-Seconds : 96.529228
H4
on second one
----------ACP FPGA EXECUTION TIME--------------------
Execution Time for ACP FPGA in Micro-Seconds : 96.747696
Console
TCF Debug Virtual Terminal - ARM
Search
ENG IN
13:07
03-04-2023