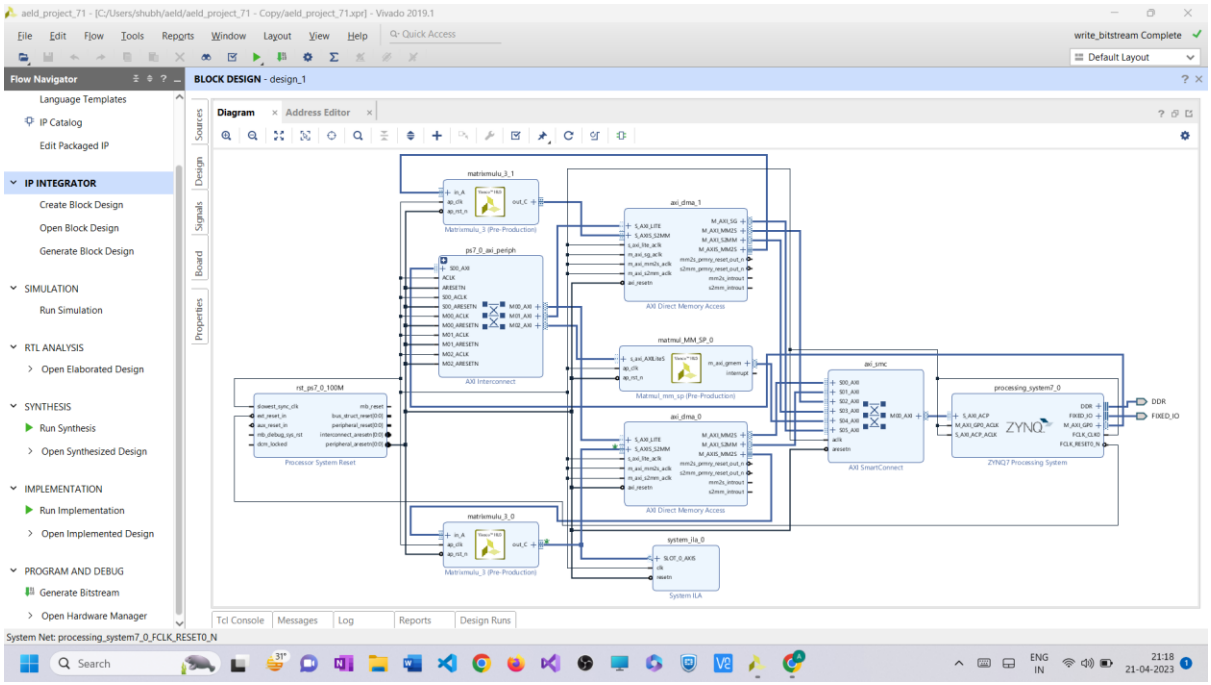
Name – Aryan Gupta

Roll No -MT22154

Lab no -11

Advanced Embedded Logic Design

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 "xmatmul_mm_sp.h"
#define INP_SIZE 8
#define MATSIZE 8
#define MEM_BASE_ADDR       0x01000000
#define TX_BD_SPACE_BASE    (MEM_BASE_ADDR)
#define RX_BD_SPACE_BASE    (MEM_BASE_ADDR + 0x00001000)
#define RX_BUFFER_BASE      (MEM_BASE_ADDR + 0x00300000)
int SGDMA_RxSetup(XAxiDma * AxiDmaInstPtr, u32 No_of_BDs);
```

```
int SGDMA_TxSetup(XAxiDma * AxiDmaInstPtr, u32 No_of_BDs);
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 DMA_input1[INP_SIZE*INP_SIZE*2];
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 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_input1[index]=Find_input_A[row][col];
                    index++;
                }
            }
            for(int row=0; row<MATSIZE; row++)
            {
                for(int col=0; col<MATSIZE; col++)
                {
                    DMA_input1[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_input1,
(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()
{
    int status;
    float DMA_input[MATSIZE*MATSIZE*2];
    u32 No_of_BDs=1;
    XAxiDma AxiDMAsg;
    XAxiDma_Config *DMA_confptrsg;
    DMA_confptrsg = XAxiDma_LookupConfig(XPAR_AXI_DMA_1_DEVICE_ID);
    status = XAxiDma_CfgInitialize(&AxiDMAsg, DMA_confptrsg);
    if(status != XST_SUCCESS)
    {
        printf("DMA SG Init Failed\t\n");
        return XST_FAILURE;
    }
    if(!XAxiDma_HasSg(&AxiDMAsg))
    {
        xil_printf("Device configured as Simple mode \r\n");
        return XST_FAILURE;
    }
    status = SGDMA_TxSetup(&AxiDMAsg, No_of_BDs);
    if (status != XST_SUCCESS)
    {
        return XST_FAILURE;
    }
    status = SGDMA_RxSetup(&AxiDMAsg, No_of_BDs);
    if (status != XST_SUCCESS)
    {
        return XST_FAILURE;
    }
        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=index+1;
            }
        }
        for(int row=0;row<MATSIZE;row++)
        {
            for(int col=0;col<MATSIZE;col++)
            {
                DMA_input[index]=Find_input_B[row][col];
                index=index+1;
            }
        }
        XAxiDma_BdRing *TxRingPtr;
        XAxiDma_BdRing *RxRingPtr;
```

```
        XAxiDma_Bd *TxBdPtr, *RxBdPtr;
        RxRingPtr = XAxiDma_GetRxRing(&AxiDMAsg);
        status = XAxiDma_BdRingAlloc(RxRingPtr, 1, &RxBdPtr);
        if (status != XST_SUCCESS) {
            xil_printf("RX alloc BD failed %d\r\n", status);
            return XST_FAILURE;
        }
        UINTPTR RxBufferPtr;
        RxBufferPtr = RX_BUFFER_BASE;
        status = XAxiDma_BdSetBufAddr(RxBdPtr, RxBufferPtr);
        if (status != XST_SUCCESS)
        {
            xil_printf("Set buffer addr %x on BD %x failed %d\r\n",
                (unsigned int)RxBufferPtr, (UINTPTR)RxBdPtr, status);
            return XST_FAILURE;
        }
        status = XAxiDma_BdSetLength(RxBdPtr, (sizeof(float)*MATSIZE*MATSIZE),
                RxRingPtr->MaxTransferLen);
        if (status != XST_SUCCESS)
        {
            xil_printf("Rx set length %d on BD %x failed %d\r\n",
                    (sizeof(float)*MATSIZE*MATSIZE), (UINTPTR)RxBdPtr,
status);
            return XST_FAILURE;
        }
        XAxiDma_BdSetCtrl(RxBdPtr, 0);
        XAxiDma_BdSetId(RxBdPtr, RxBufferPtr);
        memset((void *)RX_BUFFER_BASE, 0, (sizeof(float)*MATSIZE*MATSIZE));
        TxRingPtr = XAxiDma_GetTxRing(&AxiDMAsg);
        status = XAxiDma_BdRingAlloc(TxRingPtr, 1, &TxBdPtr);
        if (status != XST_SUCCESS)
        {
            return XST_FAILURE;
        }
        status = XAxiDma_BdSetBufAddr(TxBdPtr, (UINTPTR) DMA_input);
        if (status != XST_SUCCESS)
        {
            xil_printf("Tx set buffer addr %x on BD %x failed
%d\r\n",(UINTPTR)DMA_input, (UINTPTR)TxBdPtr, status);
            return XST_FAILURE;
        }
        status = XAxiDma_BdSetLength(TxBdPtr,
(sizeof(float)*MATSIZE*MATSIZE*2),TxRingPtr->MaxTransferLen);
        if (status != XST_SUCCESS)
        {
            xil_printf("Tx set length %d on BD %x failed
%d\r\n",(sizeof(float)*MATSIZE*MATSIZE*2), (UINTPTR)TxBdPtr, status);
            return XST_FAILURE;
```

```
        }
        XAxiDma_BdSetCtrl(TxBdPtr, XAXIDMA_BD_CTRL_TXEOF_MASK |
XAXIDMA_BD_CTRL_TXSOF_MASK);
        XAxiDma_BdSetId(TxBdPtr, (UINTPTR)DMA_input);
        int Txcount,Rxcount;
        XTime time_PL_start , time_PL_end;
        XTime_SetTime(0);
        XTime_GetTime(&time_PL_start);
        status = XAxiDma_BdRingToHw(RxRingPtr, 1, RxBdPtr);
        if (status != XST_SUCCESS)
        {
            xil_printf("RX submit hw failed %d\r\n", status);
            return XST_FAILURE;
        }
        status = XAxiDma_BdRingToHw(TxRingPtr, 1, TxBdPtr);
        if (status != XST_SUCCESS) {
            xil_printf("to hw failed %d\r\n", status);
            return XST_FAILURE;
        }
        while(!(XAxiDma_BdRead(TxBdPtr,
XAXIDMA_BD_STS_OFFSET)&XAXIDMA_BD_STS_COMPLETE_MASK));
        while(!(XAxiDma_BdRead(RxBdPtr,
XAXIDMA_BD_STS_OFFSET)&XAXIDMA_BD_STS_COMPLETE_MASK));
        XTime_GetTime(&time_PL_end);
        Txcount = XAxiDma_BdRingFromHw(TxRingPtr,1, &TxBdPtr);
        Rxcount = XAxiDma_BdRingFromHw(RxRingPtr,1, &RxBdPtr);
        status = XAxiDma_BdRingFree(TxRingPtr, Txcount, TxBdPtr);
        if (status != XST_SUCCESS) {
            xil_printf("Failed to free %d tx BDs %d\r\n",Txcount, status);
            return XST_FAILURE;
        }
        status = XAxiDma_BdRingFree(RxRingPtr, Rxcount, RxBdPtr);
        if (status != XST_SUCCESS) {
            xil_printf("Failed to free %d rx BDs %d\r\n",Rxcount, status);
            return XST_FAILURE;
        }
        float time_w = ((float)1.0 * (time_PL_end - time_PL_start) /
(COUNTS_PER_SECOND/1000000));
        printf("Execution Time for for SG DMA based PL: %f\n",time_w);
        index =0;
        float *DMA_output2 = (float *) RX_BUFFER_BASE;
        for(int row=0;row<MATSIZE;row++)
        {
            for(int col=0;col<MATSIZE;col++)
            {
                Find_output2[row][col] = DMA_output2[index];
                index = index+1;
            }
```

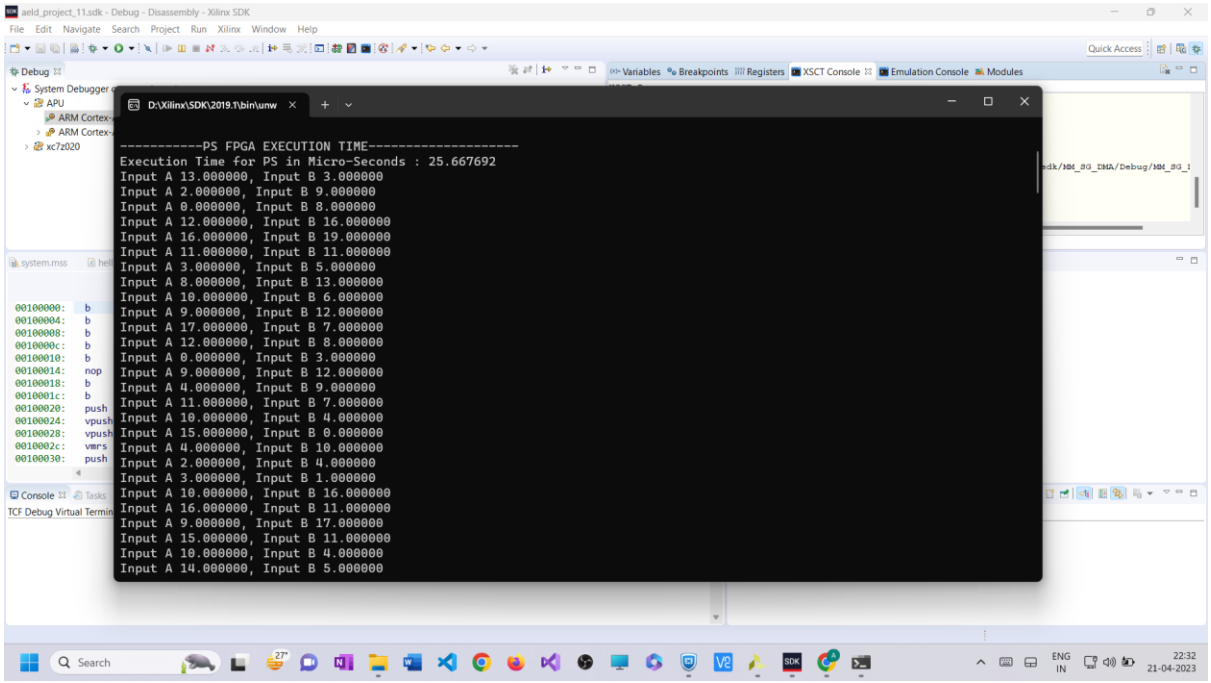
```
        }
      return 0;
}
int Find_ACP3()
{
    float Find_output_DMA[INP_SIZE*INP_SIZE];
    XMatmul_mm_sp AxiMM1;
    XMatmul_mm_sp_Config *MM_confptr1;
    MM_confptr1 = XMatmul_mm_sp_LookupConfig(XPAR_MATMUL_MM_SP_0_DEVICE_ID);
    int status = XMatmul_mm_sp_CfgInitialize(&AxiMM1,MM_confptr1);
    XTime time_PL_start , time_PL_end;
    XTime_SetTime(0);
    XTime_GetTime(&time_PL_start);
    XMatmul_mm_sp_Set_Matrix_In(&AxiMM1,(u32)DMA_input1);
    XMatmul_mm_sp_Set_Matrix_C_HW(&AxiMM1,(u32)Find_output_DMA);
    XMatmul_mm_sp_Start(&AxiMM1);
    while(XMatmul_mm_sp_IsDone(&AxiMM1)==0)
    XTime_GetTime(&time_PL_end);
    float time_MM = 0;
    time_MM = (float)1.0 * (time_PL_end - time_PL_start) /
(COUNTS_PER_SECOND/1000000);
    printf("Execution Time for AXI MM Matrix mul FPGA in Micro-Seconds : %f\n"
, time_MM);
    for(int i = 0 ; i<INP_SIZE*INP_SIZE; i++)
    {
            printf("Output  : %f\n",Find_output_DMA[i]);
    }
    int index=0;
    for(int row=0;row<MATSIZE;row++)
    {
        for(int col=0;col<MATSIZE;col++)
        {
            Find_output3[row][col] = Find_output_DMA[index];
            index = index+1;
        }
    }
     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]>0.001)
            {
                return 0;
            }
```

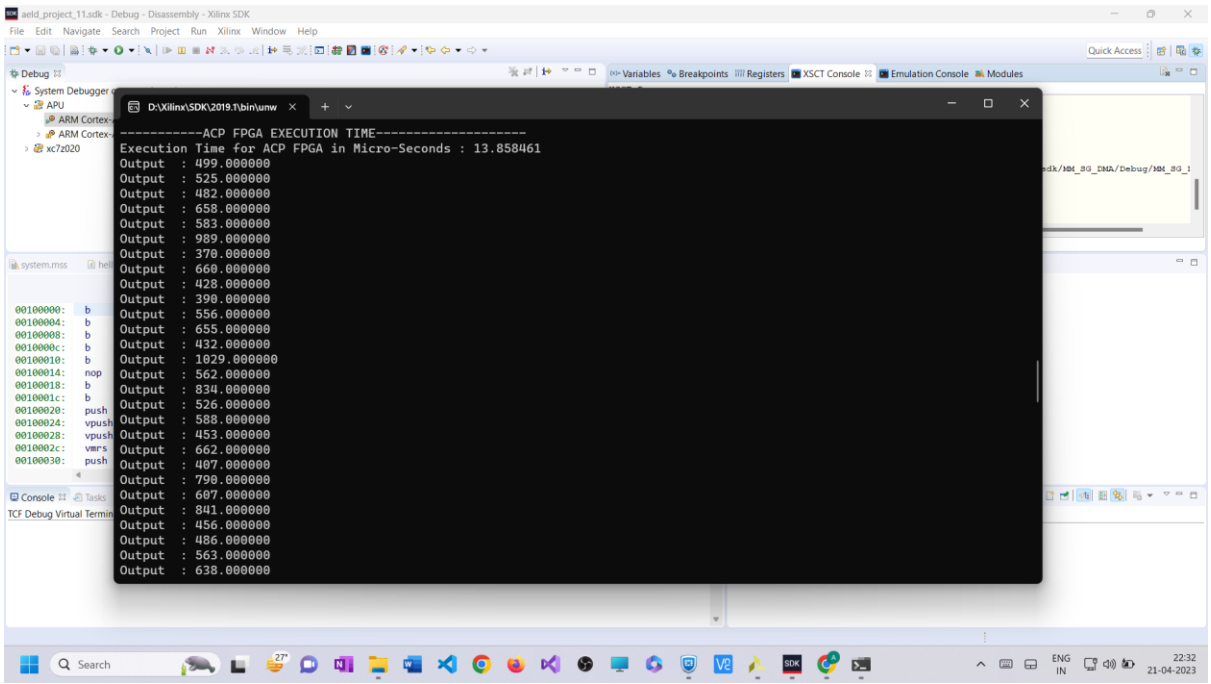
```
        }
    }
    return 1;
}
int compare2()
{
    for(int i=0; i<MATSIZE; i++)
    {
        for(int j=0; j<MATSIZE; j++)
        {
            if(Find_output2[i][j]-Find_outputps[i][j]>0.001)
            {
                return 0;
            }
        }
    }
    return 1;
}
int compare3()
{
    for(int i=0; i<MATSIZE; i++)
    {
        for(int j=0; j<MATSIZE; j++)
        {
            if(Find_output3[i][j]-Find_outputps[i][j]>0.001)
            {
                return 0;
            }
        }
    }
    return 1;
}
int main()
{
    init_platform();
    input();
    PS();
    Find_ACP1();
    Find_ACP2();
    Find_ACP3();
    int ans=compare1();
    int ans1=compare2();
    int ans2=compare3();
    if(ans==1)
    {
        printf("You are great\n");
    }
    else
```

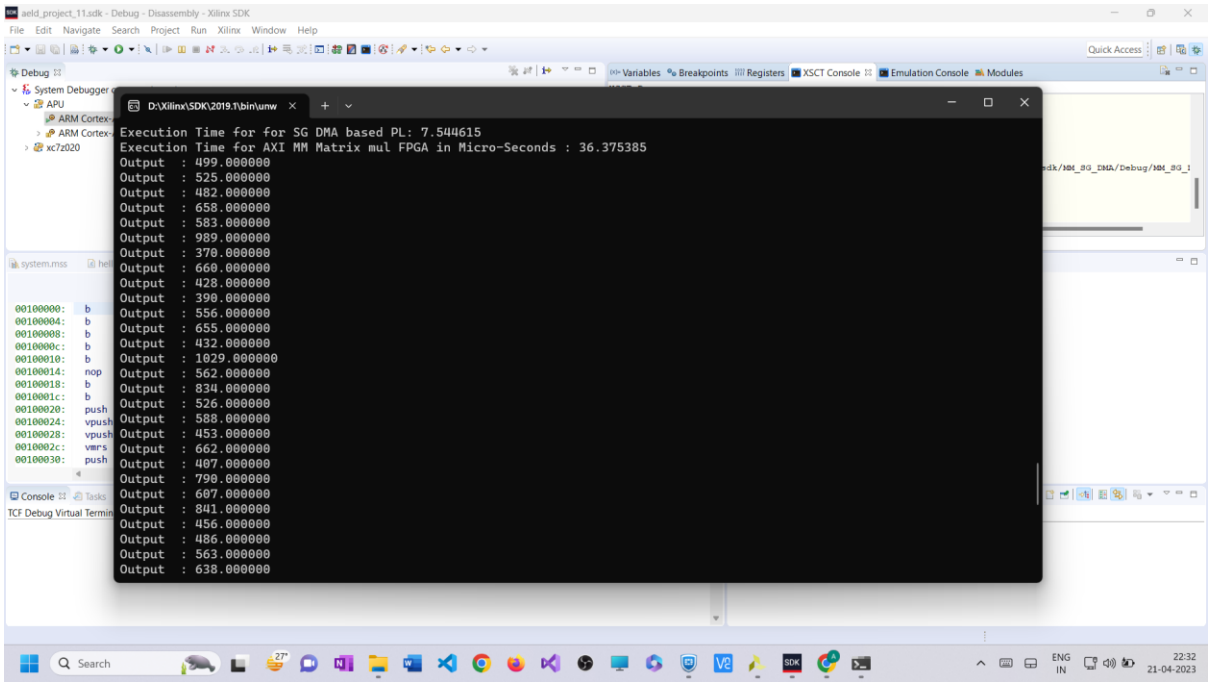
```
    {
        printf("No good at all\n");
    }
    if(ans1==1)
    {
        printf("You are great\n");
    }
    else
    {
        printf("No good at all\n");
    }
    if(ans2==1)
    {
        printf("You are great\n");
    }
    else
    {
        printf("No good at all\n");
    }
    return 0;
}
int SGDMA_RxSetup(XAxiDma * AxiDmaInstPtr, u32 No_of_BDs)
{
    XAxiDma_BdRing *RxRingPtr;
    int status;
    XAxiDma_Bd BdTemplate;
    RxRingPtr = XAxiDma_GetRxRing(AxiDmaInstPtr);
    XAxiDma_BdRingIntDisable(RxRingPtr, XAXIDMA_IRQ_ALL_MASK);
    status = XAxiDma_BdRingCreate(RxRingPtr,
RX_BD_SPACE_BASE,RX_BD_SPACE_BASE, XAXIDMA_BD_MINIMUM_ALIGNMENT, No_of_BDs);
    if (status != XST_SUCCESS) {
        xil_printf("RX create BD ring failed %d\r\n", status);
        return XST_FAILURE;
    }
    XAxiDma_BdClear(&BdTemplate);
    status = XAxiDma_BdRingClone(RxRingPtr, &BdTemplate);
    if (status != XST_SUCCESS)
    {
        xil_printf("RX clone BD failed %d\r\n", status);
        return XST_FAILURE;
    }
    status = XAxiDma_BdRingStart(RxRingPtr);
    if (status != XST_SUCCESS)
    {
        xil_printf("RX start hw failed %d\r\n", status);
        return XST_FAILURE;
    }
    return XST_SUCCESS;
```

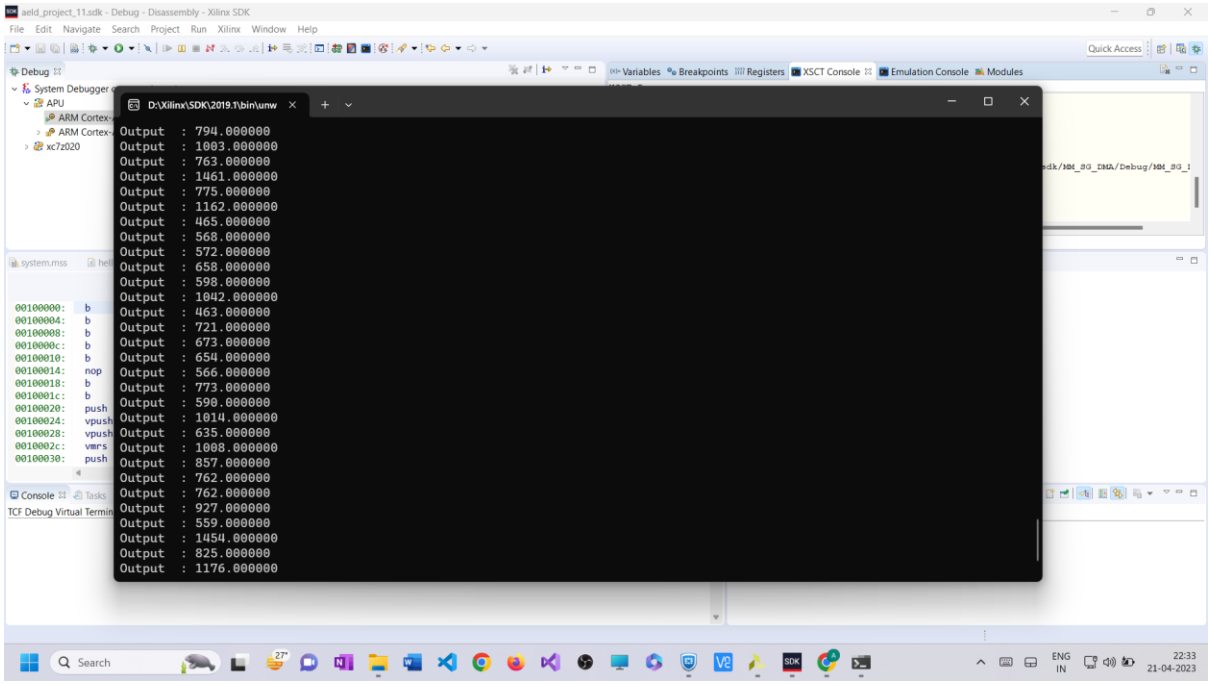
```
}
int SGDMA_TxSetup(XAxiDma * AxiDmaInstPtr, u32 No_of_BDs)
{
    XAxiDma_BdRing *TxRingPtr;
    XAxiDma_Bd BdTemplate;
    int status;
    TxRingPtr = XAxiDma_GetTxRing(AxiDmaInstPtr);
    XAxiDma_BdRingIntDisable(TxRingPtr, XAXIDMA_IRQ_ALL_MASK);
    status = XAxiDma_BdRingCreate(TxRingPtr, TX_BD_SPACE_BASE,
TX_BD_SPACE_BASE, XAXIDMA_BD_MINIMUM_ALIGNMENT, No_of_BDs);
    if (status != XST_SUCCESS)
    {
        xil_printf("failed create BD ring in txsetup\r\n");

        return XST_FAILURE;
    }
    XAxiDma_BdClear(&BdTemplate);
    status = XAxiDma_BdRingClone(TxRingPtr, &BdTemplate);
    if (status != XST_SUCCESS)
    {
        xil_printf("failed bdring clone in txsetup %d\r\n", status);
        return XST_FAILURE;
    }
    status = XAxiDma_BdRingStart(TxRingPtr);
    if (status != XST_SUCCESS)
    {
        xil_printf("failed start bdring txsetup %d\r\n", status);
        return XST_FAILURE;
    }
    return XST_SUCCESS;
}
```

Jtagterminal:-


D:\Xilinx\SDK\2019.1\bin\unw
----------PS FPGA EXECUTION TIME-------------------
Execution Time for PS in Micro-Seconds : 25.667692
Input A 13.000000, Input B 3.000000
Input A 2.000000, Input B 9.000000
Input A 0.000000, Input B 8.000000
Input A 12.000000, Input B 16.000000
Input A 16.000000, Input B 19.000000
Input A 11.000000, Input B 11.000000
Input A 3.000000, Input B 5.000000
Input A 8.000000, Input B 13.000000
Input A 10.000000, Input B 6.000000
Input A 9.000000, Input B 12.000000
Input A 17.000000, Input B 7.000000
Input A 12.000000, Input B 8.000000
Input A 0.000000, Input B 3.000000
Input A 9.000000, Input B 12.000000
Input A 4.000000, Input B 9.000000
Input A 11.000000, Input B 7.000000
Input A 10.000000, Input B 4.000000
Input A 15.000000, Input B 0.000000
Input A 4.000000, Input B 10.000000
Input A 2.000000, Input B 4.000000
Input A 3.000000, Input B 1.000000
Input A 10.000000, Input B 16.000000
Input A 16.000000, Input B 11.000000
Input A 9.000000, Input B 17.000000
Input A 15.000000, Input B 11.000000
Input A 10.000000, Input B 4.000000
Input A 14.000000, Input B 5.000000


D:\Xilinx\SDK\2019.1\bin\unw
----------ACP FPGA EXECUTION TIME--------------------
Execution Time for ACP FPGA in Micro-Seconds : 13.858461
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
Output : 588.000000
Output : 453.000000
Output : 662.000000
Output : 407.000000
Output : 790.000000
Output : 607.000000
Output : 841.000000
Output : 456.000000
Output : 486.000000
Output : 563.000000
Output : 638.000000


aeld_project_11.sdk - Debug - Disassembly - Xilinx SDK
File Edit Navigate Search Project Run Xilinx Window Help
D:\Xilinx\SDK\2019.1\bin\unw
Execution Time for for SG DMA based PL: 7.544615
Execution Time for AXI MM Matrix mul FPGA in Micro-Seconds : 36.375385
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
Output : 588.000000
Output : 453.000000
Output : 662.000000
Output : 407.000000
Output : 790.000000
Output : 607.000000
Output : 841.000000
Output : 456.000000
Output : 486.000000
Output : 563.000000
Output : 638.000000
Search
ENG IN
22:32
21-04-2023


aeld_project_11.sdk - Debug - Disassembly - Xilinx SDK
File Edit Navigate Search Project Run Xilinx Window Help
D:\Xilinx\SDK\2019.1\bin\unw
Output : 794.000000
Output : 1003.000000
Output : 763.000000
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
Search
ENG IN
22:33
21-04-2023