Lab 10

Tasks to be done in this Lab:

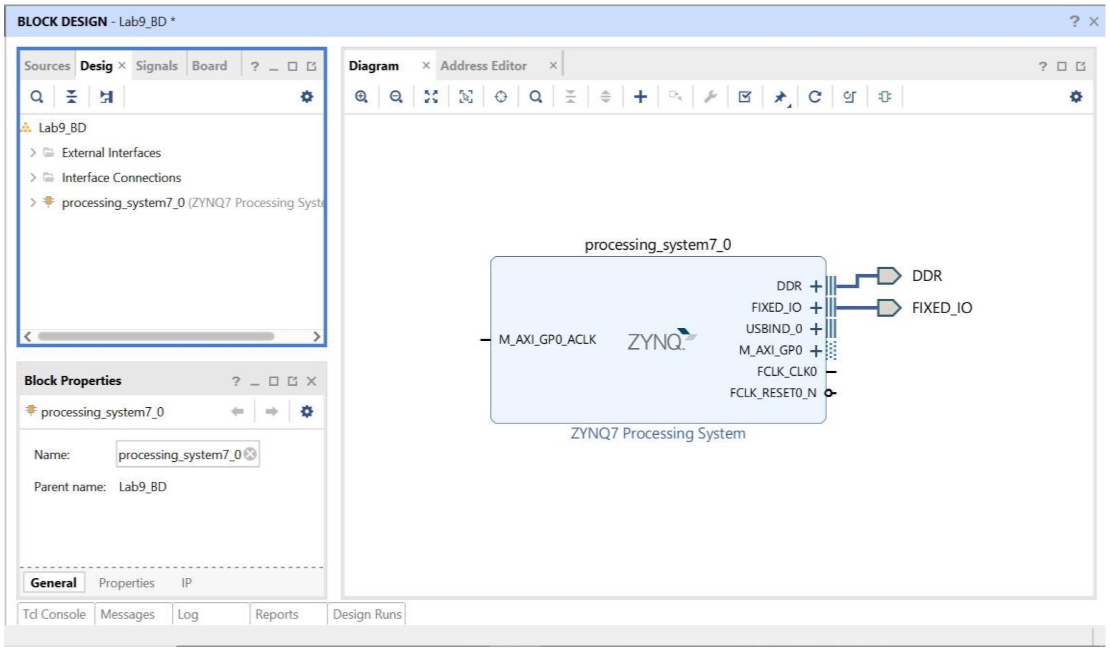
1. Create a block design in Vivado using the IP Integrator.
2. Create an application for Matrix Multiplication in SDK. Calculate execution time for different matrix sizes and observe the effect of optimizations on the execution time.
3. Learn Debugging Mode in SDK

Topics to explore: 1) Zynq Architecture, 2) Zynq Configuration, 3) Creating Project in Vivado using IP Integrator, 4) Using SDK to create C-based Application for ARM, 5) Debugging Mode

Part -1

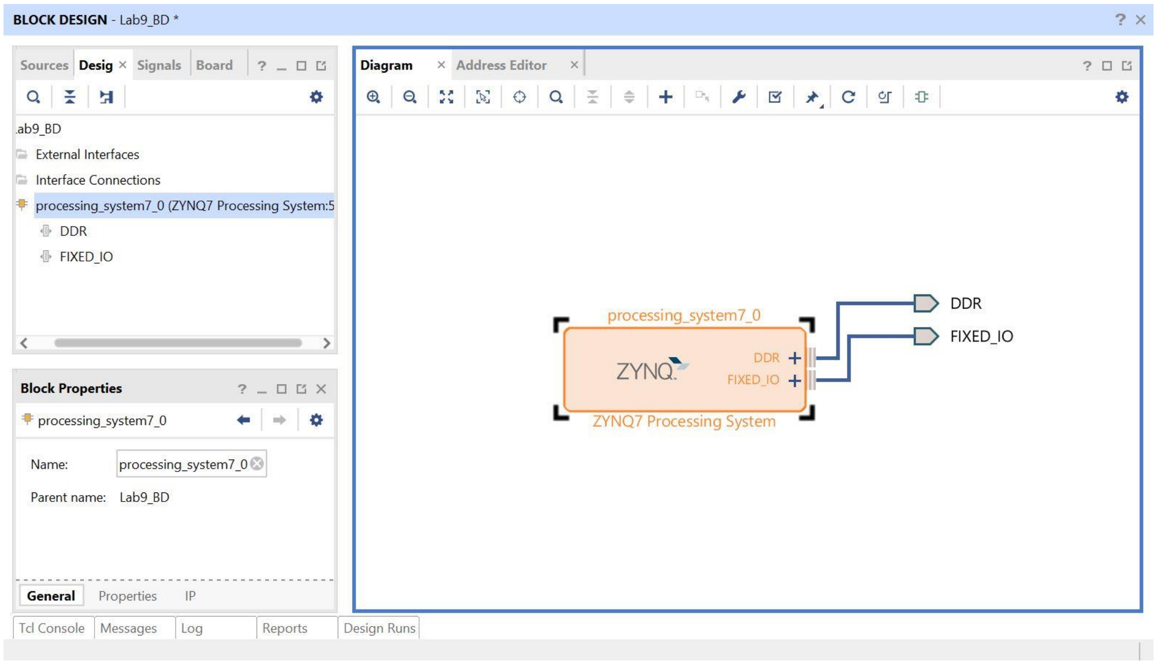
Part-1 remains the same as Lab 9. We only need the Zynq PS. PL is not required for this Lab. Please follow the following steps to reach the needed block diagram

1. Create a new Vivado Project.
2. At the next window, tick "Do not specify sources at this time."
3. In the board selection window, select Zybo Z7-10 and click on Finish in the next window.
4. Under the IP Integrator dropdown, click on Create Block Design.
5. In the diagram, window click on +(Add IP sign) and search for Zynq PS, and double click on it.
6. Once the IP got placed in the diagram window, "Run Block Automation." This step needs to be done every time you add an IP.
7. After running the block automation, your block diagram should look like the one given below:

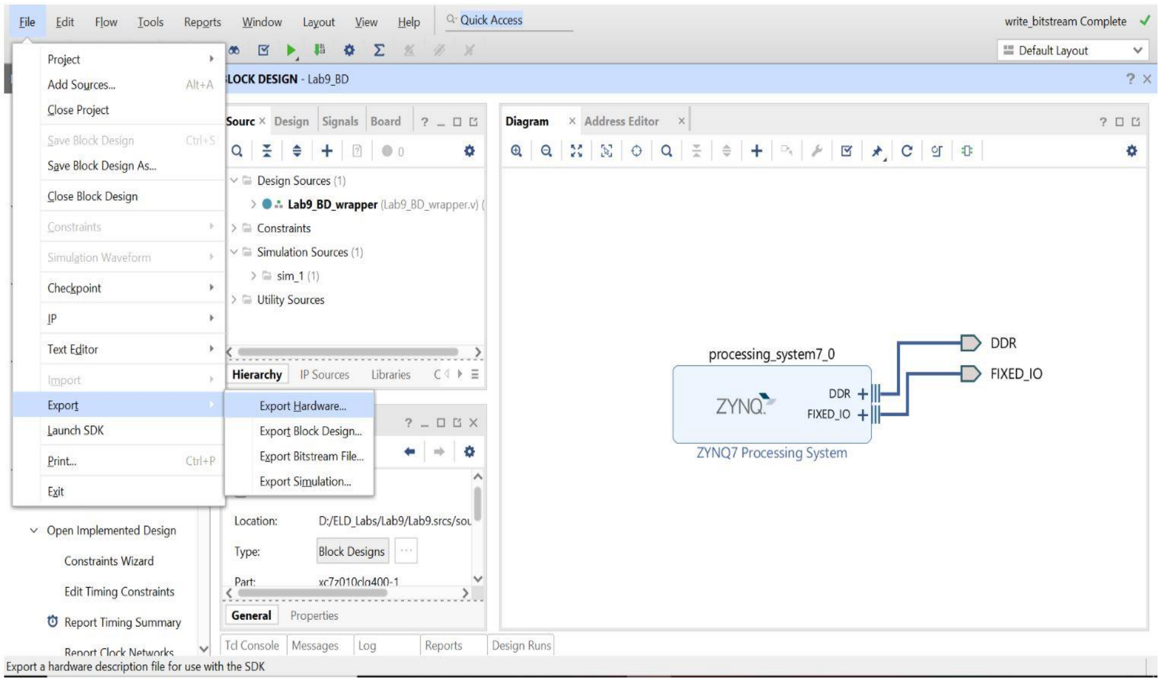


Name: processing_system7_0
Parent name: Lab9_BD

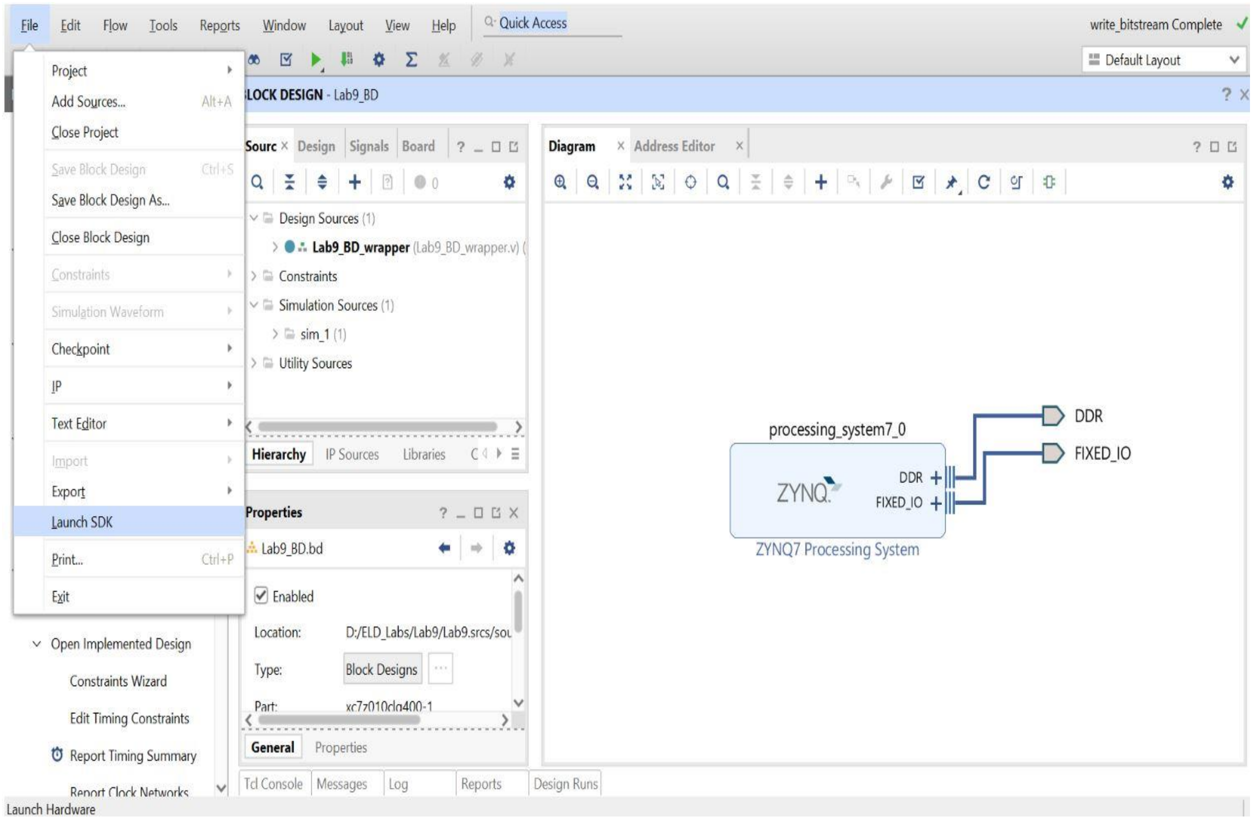
8. Customize the IP like we did in Lab 9. Your final block diagram should look like the one shown below.



9. The next three steps are important and need to be done in the same order. Validate the design, generate output product, and create HDL wrapper. DO NOT ever miss these steps.
10. Bitstream generation is optional.
11. The final step is to export hardware and Launch SDK. Follow the following screenshots to do so.

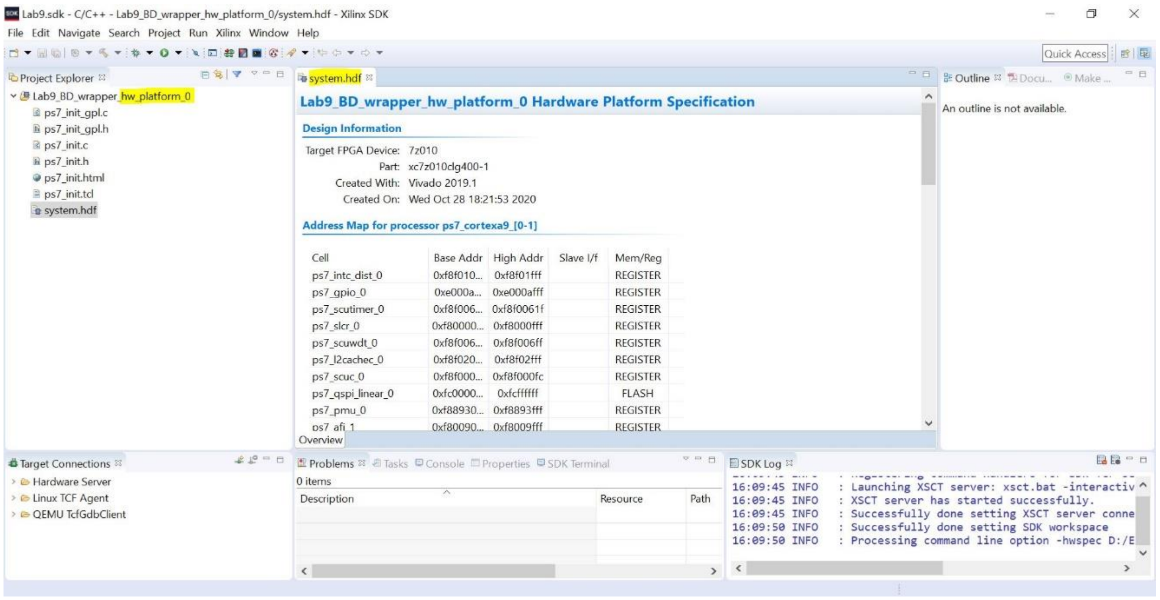


No need to include bitstream this time because we have not used the FPGA.
Finally, Launch SDK.



Part-2

1. Once the SDK is launched, you can see the hardware platform under the Project Explorer Tab, and by default, system.hdf will be open as shown below.



system.hdf

Lab9_BD_wrapper_hw_platform_0 Hardware Platform Specification

Design Information

Target FPGA Device:	7z010
Part:	xc7z010clg400-1
Created With:	Vivado 2019.1
Created On:	Wed Oct 28 18:21:53 2020

Address Map for processor ps7_cortexa9_[0-1]

Cell	Base Addr	High Addr	Slave I/f	Mem/Reg
ps7_intc_dist_0	0xf8f010...	0xf8f01fff		REGISTER
ps7_gpio_0	0xe000a...	0xe000afff		REGISTER
ps7_scutimer_0	0xf8f006...	0xf8f0061f		REGISTER
ps7_slcr_0	0xf80000...	0xf8000fff		REGISTER
ps7_scuwdt_0	0xf8f006...	0xf8f006ff		REGISTER
ps7_l2cachec_0	0xf8f020...	0xf8f02fff		REGISTER
ps7_scuc_0	0xf8f000...	0xf8f000fc		REGISTER
ps7_qspi_linear_0	0xfc0000...	0xfcffffff		FLASH
ps7_pmu_0	0xf88930...	0xf8893fff		REGISTER
ps7_afi_1	0xf80090...	0xf8009fff		REGISTER

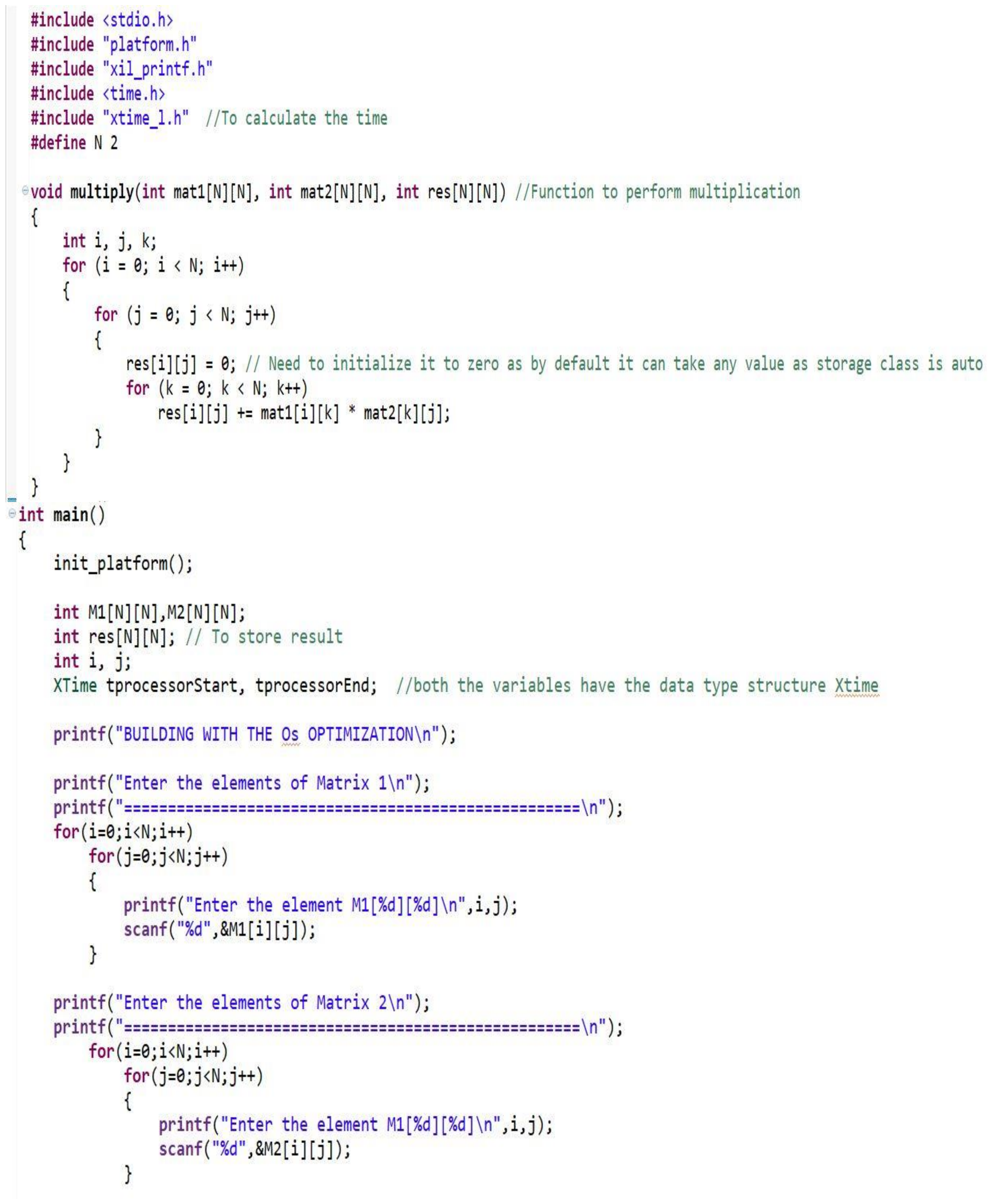
0 items

Description	Resource	Path

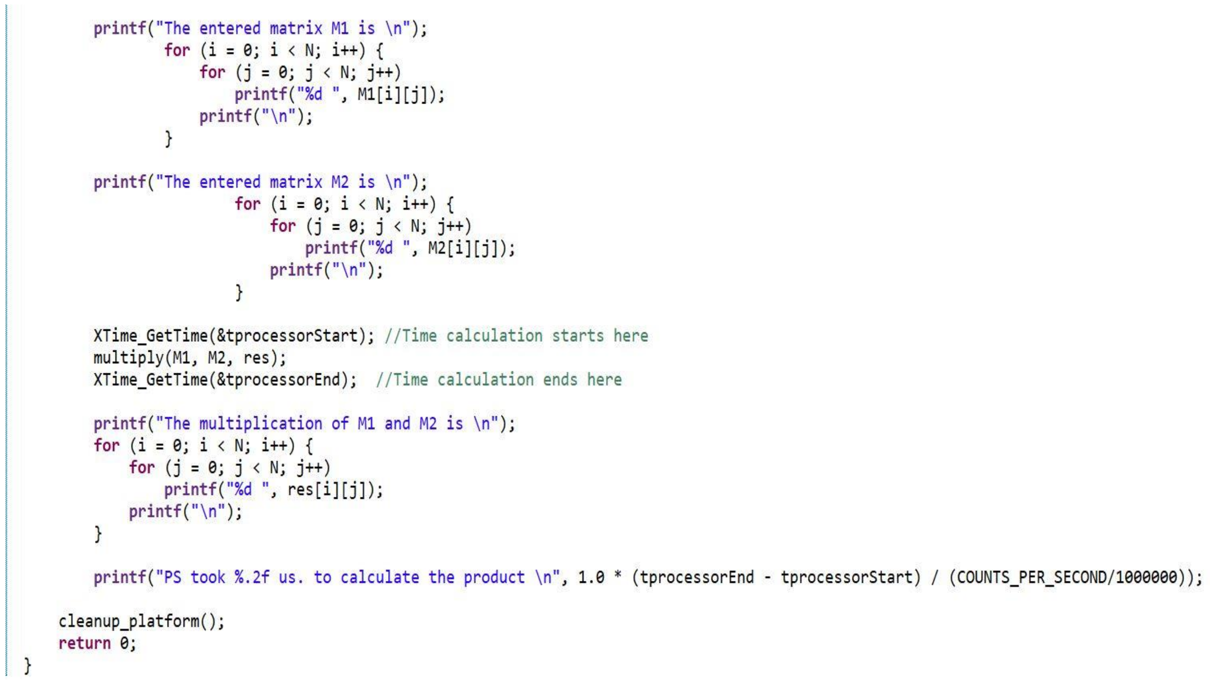
```
16:09:45 INFO   : Launching XSCT server: xsct.bat -interactiv
16:09:45 INFO   : XSCT server has started successfully.
16:09:45 INFO   : Successfully done setting XSCT server conne
16:09:50 INFO   : Successfully done setting SDK workspace
16:09:50 INFO   : Processing command line option -hwspec D:/E
```

2. Next, we will create an application project. For that, go to File->New->Application Project. Select the project's name, keep the target language as C. Don't click on Finish instead, click on next, and select the Hello World Project.

3. Navigate to the source code(.c file) of the application and make the following changes in the default code.

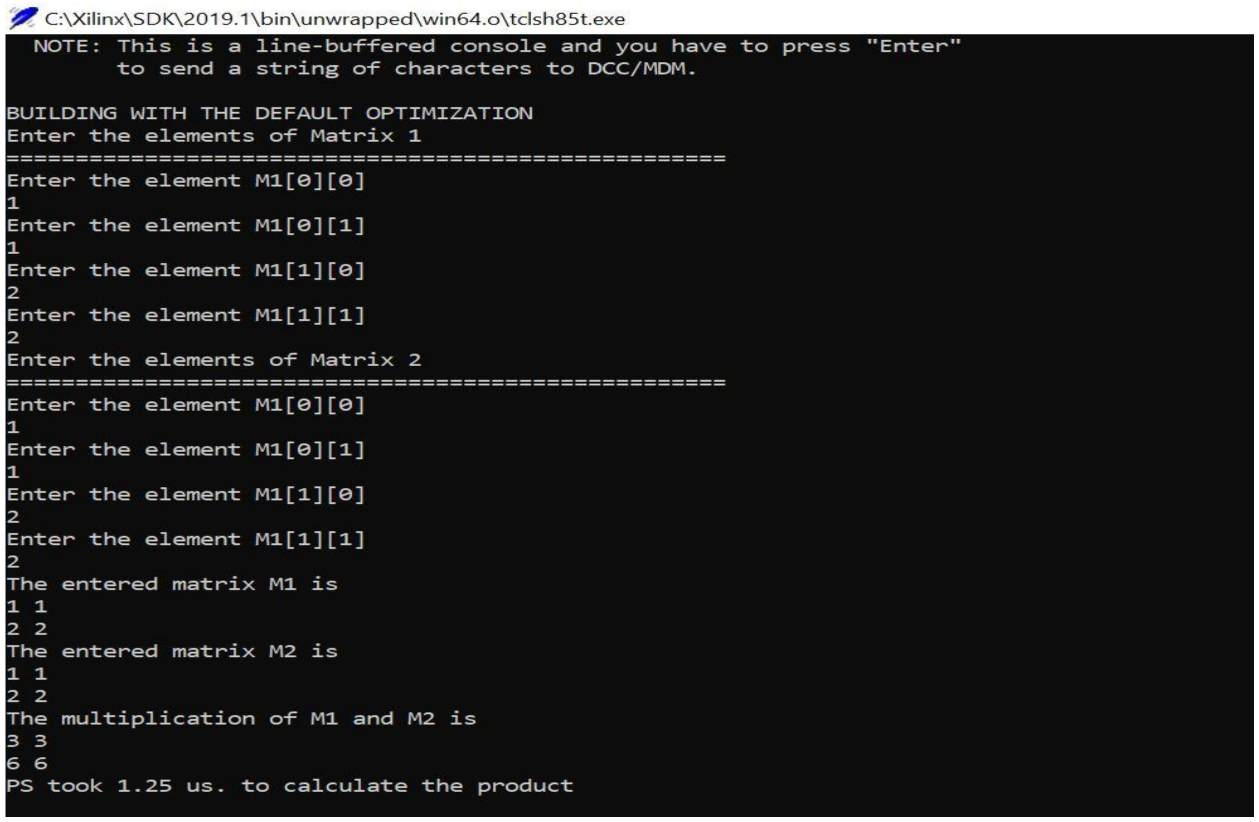


```
#include <stdio.h>
#include "platform.h"
#include "xil_printf.h"
#include <time.h>
#include "xtime_l.h"  //To calculate the time
#define N 2

void multiply(int mat1[N][N], int mat2[N][N], int res[N][N]) //Function to perform multiplication
{
    int i, j, k;
    for (i = 0; i < N; i++)
    {
        for (j = 0; j < N; j++)
        {
            res[i][j] = 0; // Need to initialize it to zero as by default it can take any value as storage class is auto
            for (k = 0; k < N; k++)
                res[i][j] += mat1[i][k] * mat2[k][j];
        }
    }
}
int main()
{
    init_platform();

    int M1[N][N],M2[N][N];
    int res[N][N]; // To store result
    int i, j;
    XTime tprocessorStart, tprocessorEnd;  //both the variables have the data type structure Xtime

    printf("BUILDING WITH THE Os OPTIMIZATION\n");

    printf("Enter the elements of Matrix 1\n");
    printf("===================================================\n");
    for(i=0;i<N;i++)
        for(j=0;j<N;j++)
        {
            printf("Enter the element M1[%d][%d]\n",i,j);
            scanf("%d",&M1[i][j]);
        }

    printf("Enter the elements of Matrix 2\n");
    printf("===================================================\n");
        for(i=0;i<N;i++)
            for(j=0;j<N;j++)
            {
                printf("Enter the element M1[%d][%d]\n",i,j);
                scanf("%d",&M2[i][j]);
            }
```



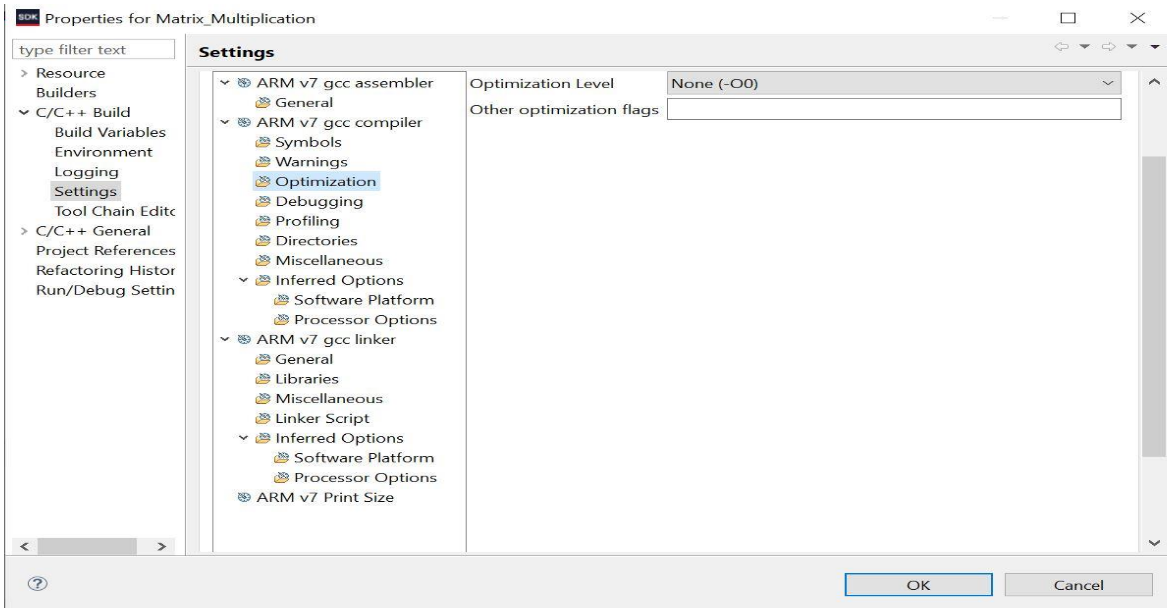
```
        printf("The entered matrix M1 is \n");
                for (i = 0; i < N; i++) {
                    for (j = 0; j < N; j++)
                        printf("%d ", M1[i][j]);
                    printf("\n");
                }

        printf("The entered matrix M2 is \n");
                        for (i = 0; i < N; i++) {
                            for (j = 0; j < N; j++)
                                printf("%d ", M2[i][j]);
                            printf("\n");
                        }

        XTime_GetTime(&tprocessorStart); //Time calculation starts here
        multiply(M1, M2, res);
        XTime_GetTime(&tprocessorEnd);  //Time calculation ends here

        printf("The multiplication of M1 and M2 is \n");
        for (i = 0; i < N; i++) {
            for (j = 0; j < N; j++)
                printf("%d ", res[i][j]);
            printf("\n");
        }

        printf("PS took %.2f us. to calculate the product \n", 1.0 * (tprocessorEnd - tprocessorStart) / (COUNTS_PER_SECOND/1000000));

    cleanup_platform();
    return 0;
}
```

4. Since we are remotely accessing the board, we will use JTAG Terminal for standard Input and Output. Add the target, configure the STDIN and STDOUT in BSP, and launch the debug configurations as we did in Lab-9.
5. Run the application after launching the JTAG Terminal. Provide the input for matrix elements. The output is shown below.

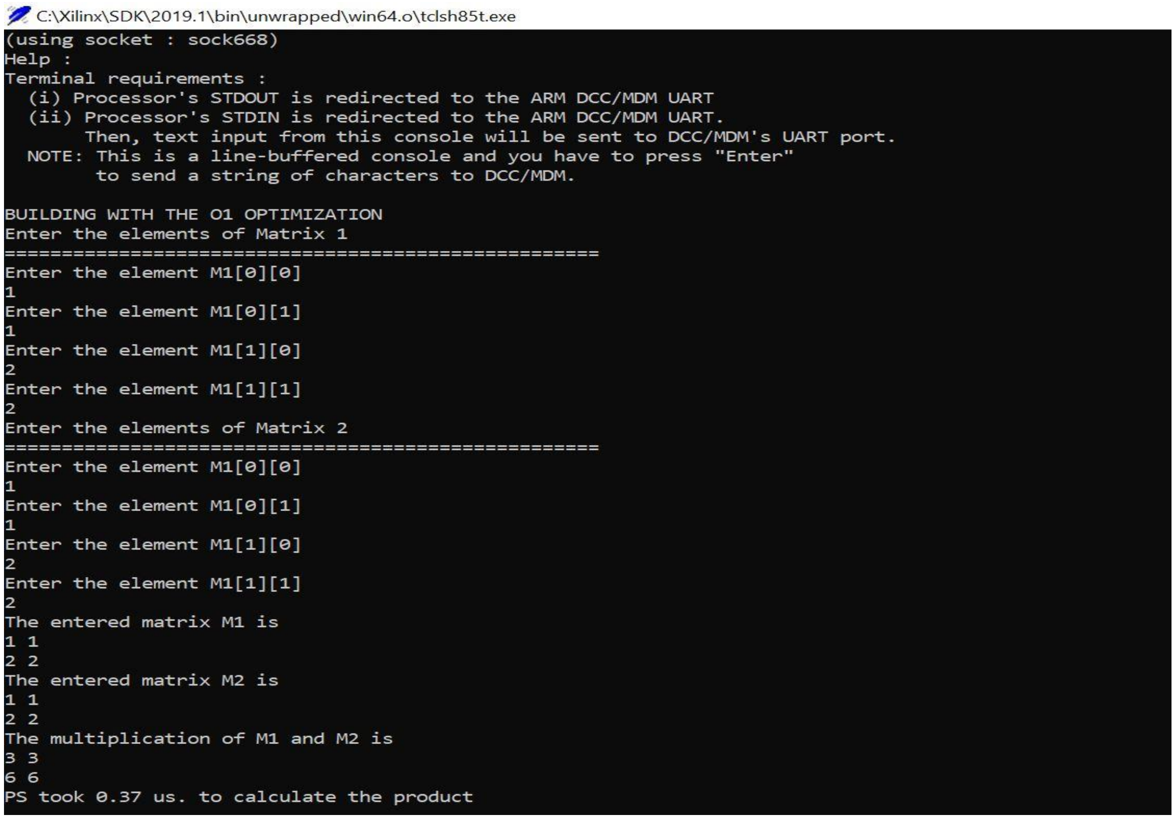


```
C:\Xilinx\SDK\2019.1\bin\unwrapped\win64.o\tclsh85t.exe
  NOTE: This is a line-buffered console and you have to press "Enter"
        to send a string of characters to DCC/MDM.

BUILDING WITH THE DEFAULT OPTIMIZATION
Enter the elements of Matrix 1
====================================================
Enter the element M1[0][0]
1
Enter the element M1[0][1]
1
Enter the element M1[1][0]
2
Enter the element M1[1][1]
2
Enter the elements of Matrix 2
====================================================
Enter the element M1[0][0]
1
Enter the element M1[0][1]
1
Enter the element M1[1][0]
2
Enter the element M1[1][1]
2
The entered matrix M1 is
1 1
2 2
The entered matrix M2 is
1 1
2 2
The multiplication of M1 and M2 is
3 3
6 6
PS took 1.25 us. to calculate the product
```

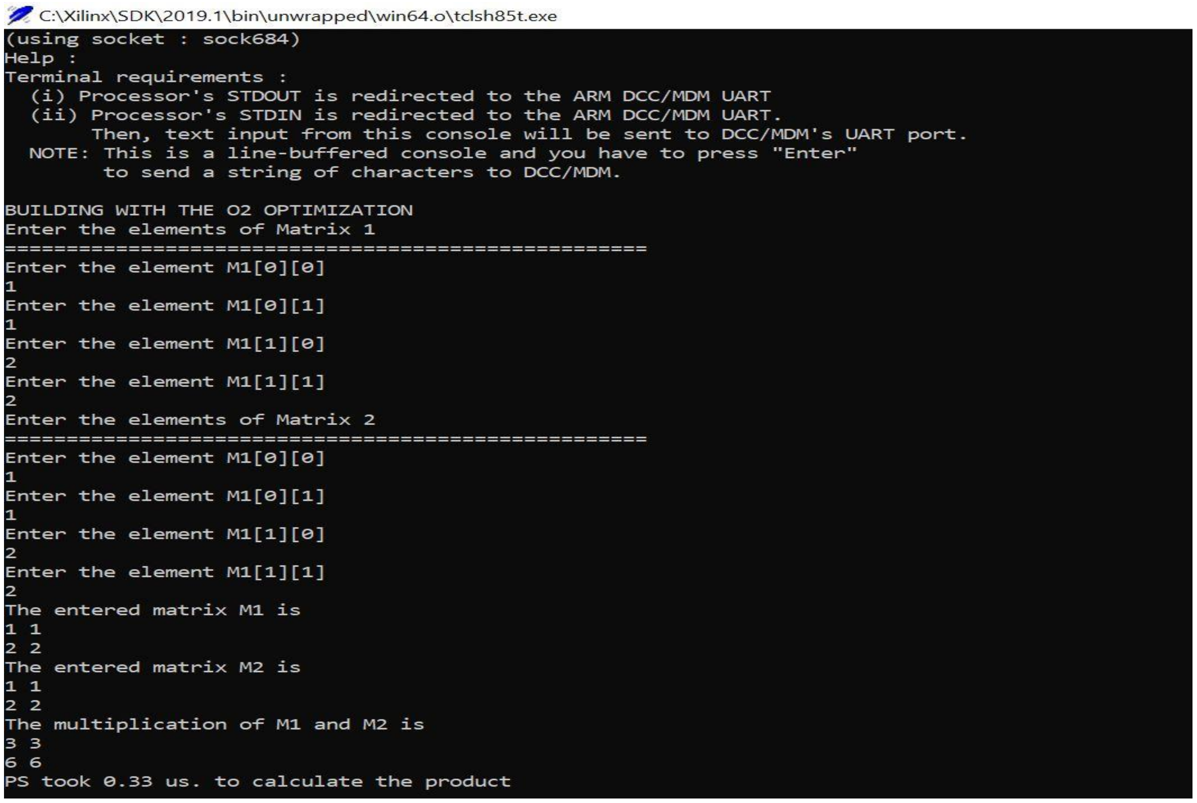
6. Next, we need to change the compiler optimizations. Right-click on the project folder, go to C/C++ build settings, then go to Optimizations. Change the optimization levels to O1, O2, O3, and Os one by one, execute the application, and observe the effect on execution time. Try to find out why is the execution time changing.



Optimization Level: None (-O0)
Other optimization flags:

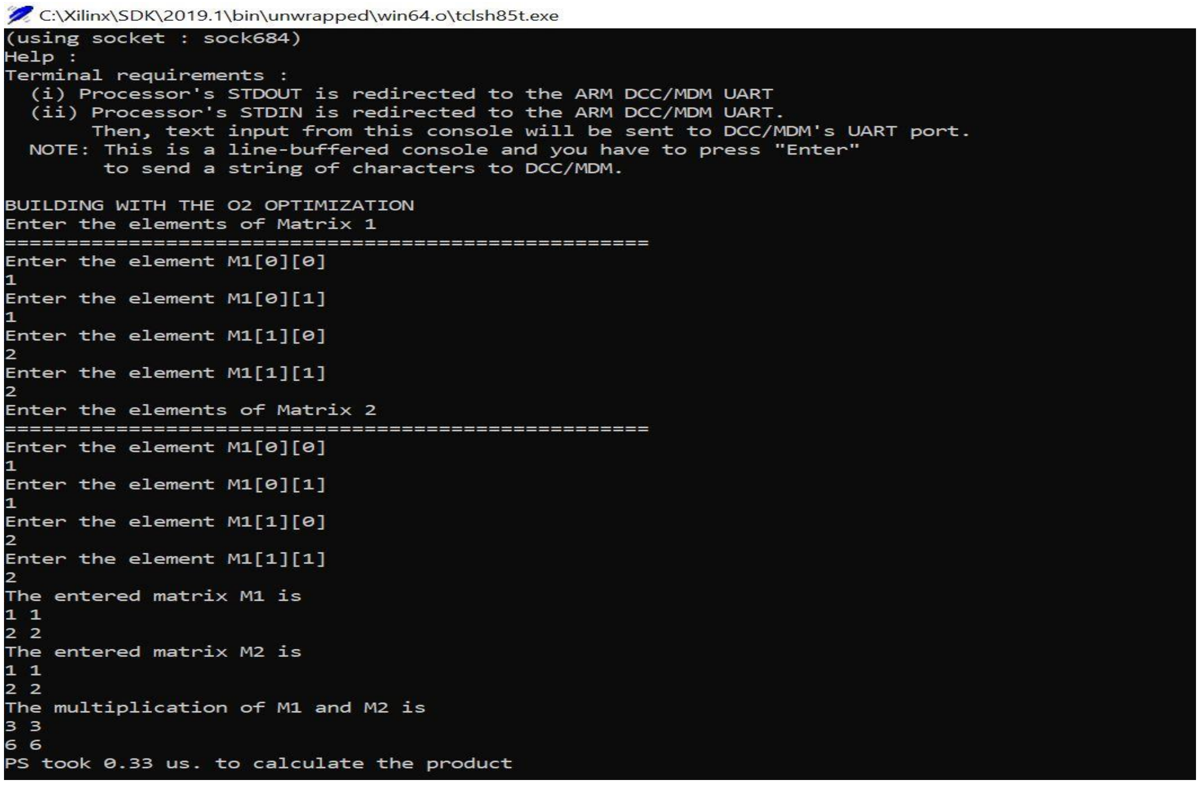

C:\Xilinx\SDK\2019.1\bin\unwrapped\win64.o\tclsh85t.exe

```
(using socket : sock668)
Help :
Terminal requirements :
  (i) Processor's STDOUT is redirected to the ARM DCC/MDM UART
  (ii) Processor's STDIN is redirected to the ARM DCC/MDM UART.
       Then, text input from this console will be sent to DCC/MDM's UART port.
  NOTE: This is a line-buffered console and you have to press "Enter"
        to send a string of characters to DCC/MDM.

BUILDING WITH THE O1 OPTIMIZATION
Enter the elements of Matrix 1
====================================================
Enter the element M1[0][0]
1
Enter the element M1[0][1]
1
Enter the element M1[1][0]
2
Enter the element M1[1][1]
2
Enter the elements of Matrix 2
====================================================
Enter the element M1[0][0]
1
Enter the element M1[0][1]
1
Enter the element M1[1][0]
2
Enter the element M1[1][1]
2
The entered matrix M1 is
1 1
2 2
The entered matrix M2 is
1 1
2 2
The multiplication of M1 and M2 is
3 3
6 6
PS took 0.37 us. to calculate the product
```



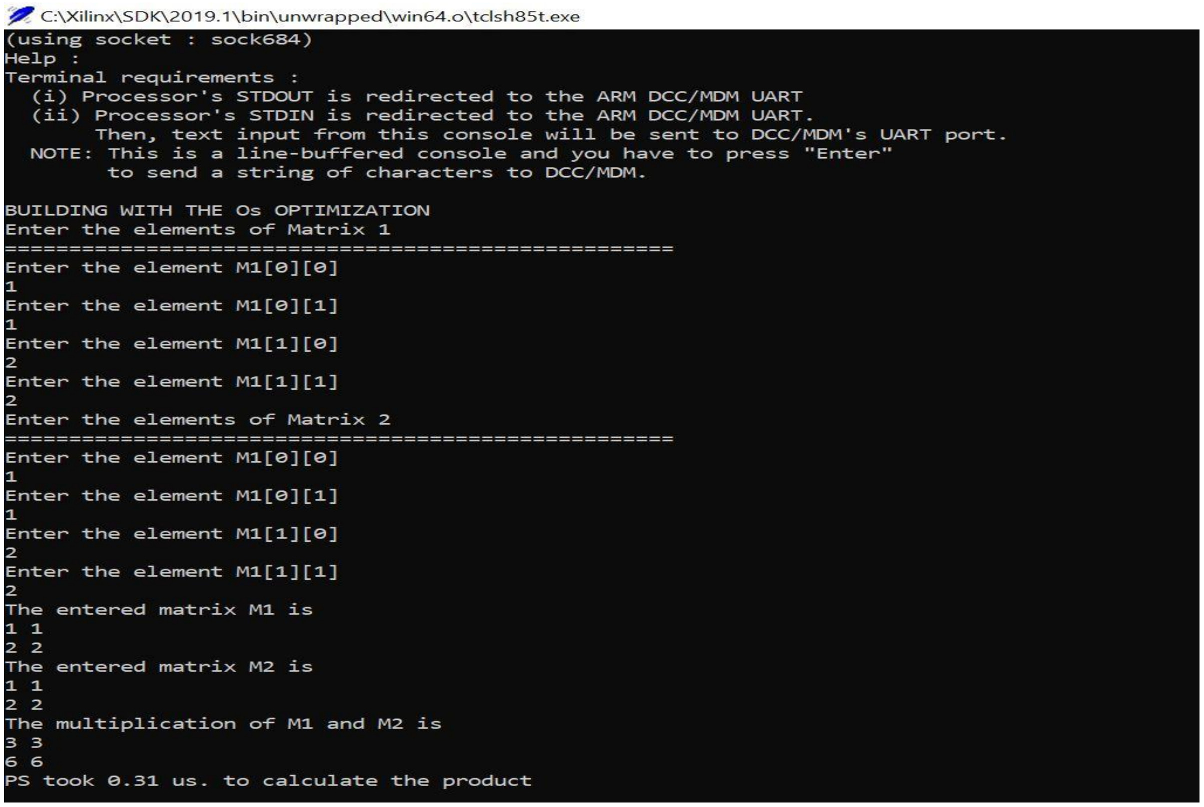
C:\Xilinx\SDK\2019.1\bin\unwrapped\win64.o\tclsh85t.exe

```
(using socket : sock684)
Help :
Terminal requirements :
  (i) Processor's STDOUT is redirected to the ARM DCC/MDM UART
  (ii) Processor's STDIN is redirected to the ARM DCC/MDM UART.
       Then, text input from this console will be sent to DCC/MDM's UART port.
  NOTE: This is a line-buffered console and you have to press "Enter"
        to send a string of characters to DCC/MDM.

BUILDING WITH THE O2 OPTIMIZATION
Enter the elements of Matrix 1
====================================================
Enter the element M1[0][0]
1
Enter the element M1[0][1]
1
Enter the element M1[1][0]
2
Enter the element M1[1][1]
2
Enter the elements of Matrix 2
====================================================
Enter the element M1[0][0]
1
Enter the element M1[0][1]
1
Enter the element M1[1][0]
2
Enter the element M1[1][1]
2
The entered matrix M1 is
1 1
2 2
The entered matrix M2 is
1 1
2 2
The multiplication of M1 and M2 is
3 3
6 6
PS took 0.33 us. to calculate the product
```



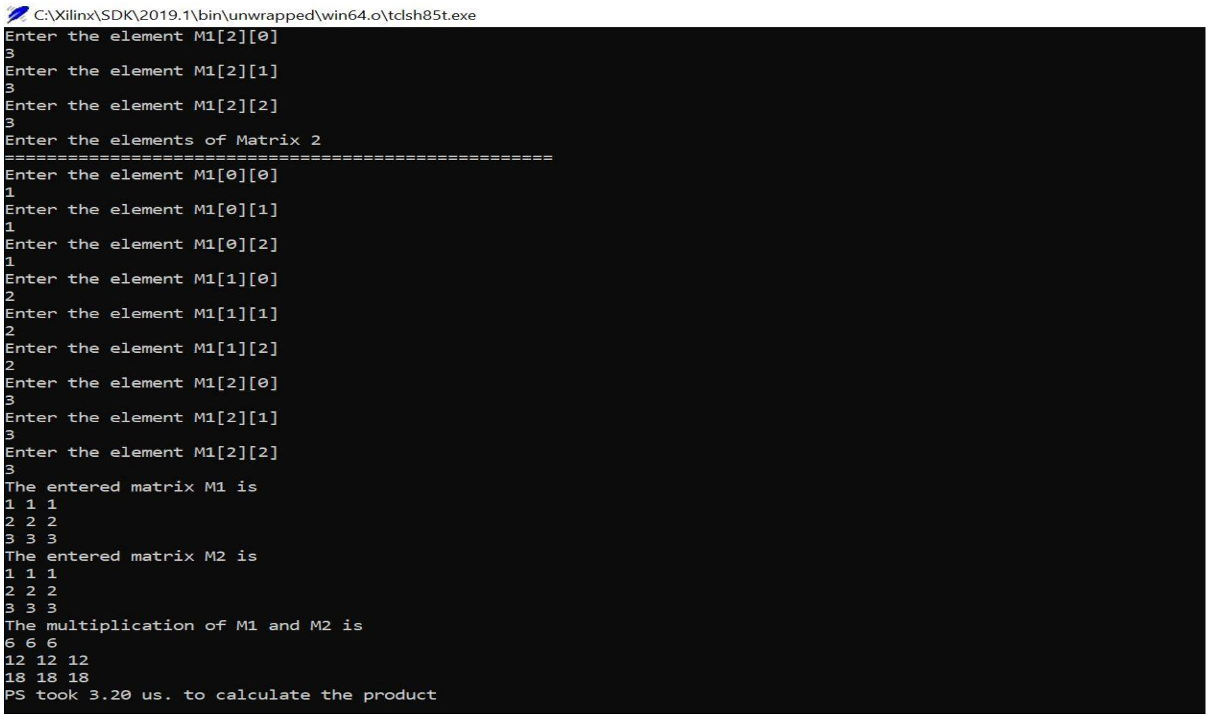
C:\Xilinx\SDK\2019.1\bin\unwrapped\win64.o\tclsh85t.exe

```
(using socket : sock684)
Help :
Terminal requirements :
  (i) Processor's STDOUT is redirected to the ARM DCC/MDM UART
  (ii) Processor's STDIN is redirected to the ARM DCC/MDM UART.
       Then, text input from this console will be sent to DCC/MDM's UART port.
  NOTE: This is a line-buffered console and you have to press "Enter"
        to send a string of characters to DCC/MDM.

BUILDING WITH THE O2 OPTIMIZATION
Enter the elements of Matrix 1
====================================================
Enter the element M1[0][0]
1
Enter the element M1[0][1]
1
Enter the element M1[1][0]
2
Enter the element M1[1][1]
2
Enter the elements of Matrix 2
====================================================
Enter the element M1[0][0]
1
Enter the element M1[0][1]
1
Enter the element M1[1][0]
2
Enter the element M1[1][1]
2
The entered matrix M1 is
1 1
2 2
The entered matrix M2 is
1 1
2 2
The multiplication of M1 and M2 is
3 3
6 6
PS took 0.33 us. to calculate the product
```



```
C:\Xilinx\SDK\2019.1\bin\unwrapped\win64.o\tclsh85t.exe
(using socket : sock684)
Help :
Terminal requirements :
  (i) Processor's STDOUT is redirected to the ARM DCC/MDM UART
  (ii) Processor's STDIN is redirected to the ARM DCC/MDM UART.
       Then, text input from this console will be sent to DCC/MDM's UART port.
  NOTE: This is a line-buffered console and you have to press "Enter"
        to send a string of characters to DCC/MDM.

BUILDING WITH THE Os OPTIMIZATION
Enter the elements of Matrix 1
====================================================
Enter the element M1[0][0]
1
Enter the element M1[0][1]
1
Enter the element M1[1][0]
2
Enter the element M1[1][1]
2
Enter the elements of Matrix 2
====================================================
Enter the element M1[0][0]
1
Enter the element M1[0][1]
1
Enter the element M1[1][0]
2
Enter the element M1[1][1]
2
The entered matrix M1 is
1 1
2 2
The entered matrix M2 is
1 1
2 2
The multiplication of M1 and M2 is
3 3
6 6
PS took 0.31 us. to calculate the product
```

7. Repeat Step-6 with different size matrices (change the order in #define statement in your code) and other optimizations and observe the effect on execution time. In the following screenshot, the code ran for a 3x3 matrix under default optimization(O0).



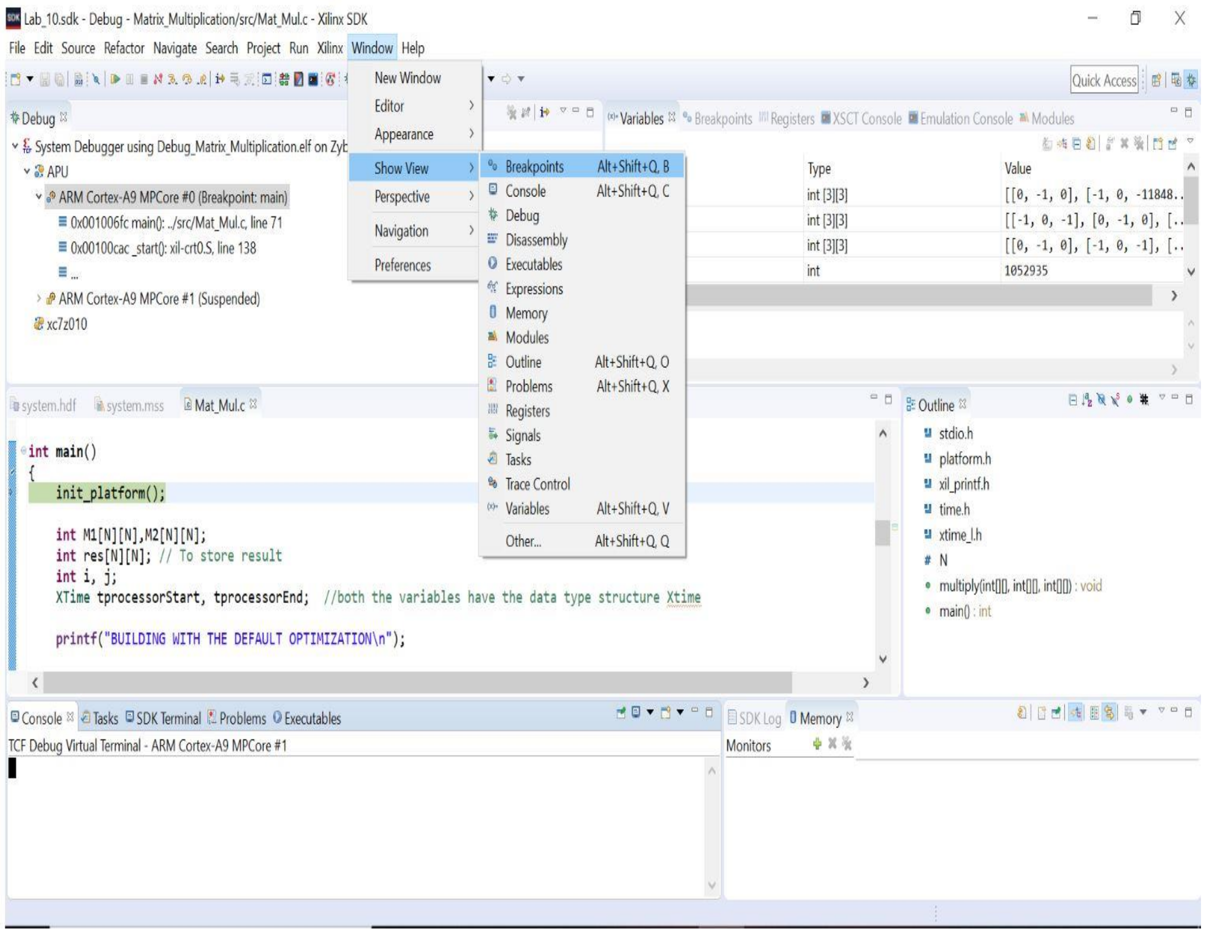
```
C:\Xilinx\SDK\2019.1\bin\unwrapped\win64.o\tclsh85t.exe
Enter the element M1[2][0]
3
Enter the element M1[2][1]
3
Enter the element M1[2][2]
3
Enter the elements of Matrix 2
====================================================
Enter the element M1[0][0]
1
Enter the element M1[0][1]
1
Enter the element M1[0][2]
1
Enter the element M1[1][0]
2
Enter the element M1[1][1]
2
Enter the element M1[1][2]
2
Enter the element M1[2][0]
3
Enter the element M1[2][1]
3
Enter the element M1[2][2]
3
The entered matrix M1 is
1 1 1
2 2 2
3 3 3
The entered matrix M2 is
1 1 1
2 2 2
3 3 3
The multiplication of M1 and M2 is
6 6 6
12 12 12
18 18 18
PS took 3.20 us. to calculate the product
```

Part-3

So far, we were launching the application using the debug perspective, but we never used the functionalities of the debugger mode. In this part, we will look into some functionalities. Using the debugger mode, we can :

- Set the breakpoints
- Monitor the values of variables
- Monitor the values of registers
- Look into the assembly code of the program.

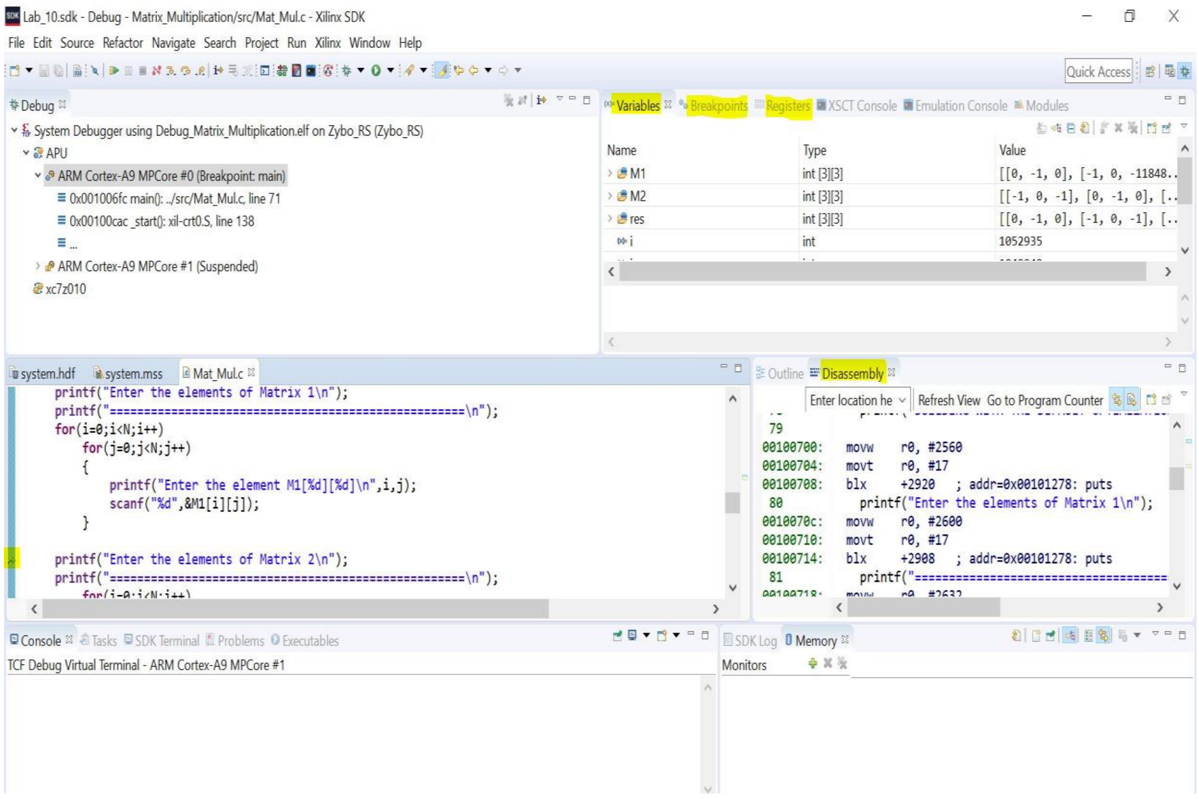
1. Launch the debugger mode using the Debug as->Debug Configurations->Debug just like we did in Lab 9.
2. If you cannot see any of the windows such as breakpoints, registers, disassembly, go to window->show view, and select the view you want to add, as shown below.


Lab_10.sdk - Debug - Matrix_Multiplication/src/Mat_Mul.c - Xilinx SDK
File Edit Source Refactor Navigate Search Project Run Xilinx Window Help
New Window
Editor
Appearance
Show View
Perspective
Navigation
Preferences
Breakpoints Alt+Shift+Q, B
Console Alt+Shift+Q, C
Debug
Disassembly
Executables
Expressions
Memory
Modules
Outline Alt+Shift+Q, O
Problems Alt+Shift+Q, X
Registers
Signals
Tasks
Trace Control
Variables Alt+Shift+Q, V
Other... Alt+Shift+Q, Q
Quick Access
Debug
System Debugger using Debug_Matrix_Multiplication.elf on Zyb
APU
ARM Cortex-A9 MPCore #0 (Breakpoint: main)
0x001006fc main(): ../src/Mat_Mul.c, line 71
0x00100cac _start(): xil-crt0.S, line 138
ARM Cortex-A9 MPCore #1 (Suspended)
xc7z010
Variables Breakpoints Registers XSCT Console Emulation Console Modules
system.hdf system.mss Mat_Mul.c
Outline
stdio.h
platform.h
xil_printf.h
time.h
xtime_l.h
N
multiply(int[][], int[][], int[][]) : void
main() : int
Console Tasks SDK Terminal Problems Executables
TCF Debug Virtual Terminal - ARM Cortex-A9 MPCore #1
SDK Log Memory
Monitors

Type	Value
int [3][3]	[[0, -1, 0], [-1, 0, -11848..
int [3][3]	[[-1, 0, -1], [0, -1, 0], [..
int [3][3]	[[0, -1, 0], [-1, 0, -1], [..
int	1052935

```
int main()
{
    init_platform();

    int M1[N][N],M2[N][N];
    int res[N][N]; // To store result
    int i, j;
    XTime tprocessorStart, tprocessorEnd;  //both the variables have the data type structure Xtime

    printf("BUILDING WITH THE DEFAULT OPTIMIZATION\n");
```

3. To add a breakpoint, double click on the blue line wherever you want to add a breakpoint. The breakpoint, disassembly, variables, and registers are highlighted in the following screenshot.



Lab_10.sdk - Debug - Matrix_Multiplication/src/Mat_Mul.c - Xilinx SDK

```
System Debugger using Debug_Matrix_Multiplication.elf on Zybo_RS (Zybo_RS)
  APU
    ARM Cortex-A9 MPCore #0 (Breakpoint: main)
      0x001006fc main(): ../src/Mat_Mul.c, line 71
      0x00100cac _start(): xil-crt0.S, line 138
    ARM Cortex-A9 MPCore #1 (Suspended)
  xc7z010
```

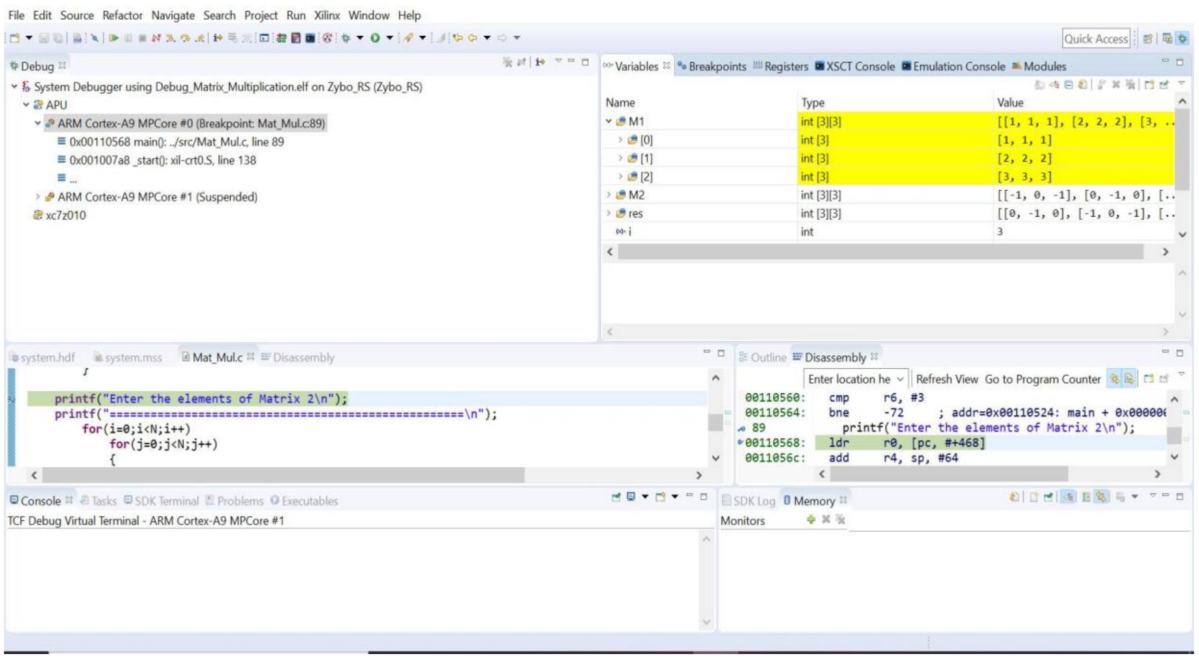
Name	Type	Value
M1	int [3][3]	[[0, -1, 0], [-1, 0, -11848..
M2	int [3][3]	[[-1, 0, -1], [0, -1, 0], [..
res	int [3][3]	[[0, -1, 0], [-1, 0, -1], [..
i	int	1052935

```
    printf("Enter the elements of Matrix 1\n");
    printf("==================================================\n");
    for(i=0;i<N;i++)
        for(j=0;j<N;j++)
        {
            printf("Enter the element M1[%d][%d]\n",i,j);
            scanf("%d",&M1[i][j]);
        }

    printf("Enter the elements of Matrix 2\n");
    printf("==================================================\n");
```

```
79
00100700:   movw    r0, #2560
00100704:   movt    r0, #17
00100708:   blx     +2920   ; addr=0x00101278: puts
80            printf("Enter the elements of Matrix 1\n");
0010070c:   movw    r0, #2600
00100710:   movt    r0, #17
00100714:   blx     +2908   ; addr=0x00101278: puts
81            printf("==================================
```

TCF Debug Virtual Terminal - ARM Cortex-A9 MPCore #1

4. Now we will monitor the value of variables when the code reaches the first breakpoint, i.e., after the user has inputted the value of matrix 1. For running, there are three options, namely resume,step-over, and step-into. Step-into will go into the code of the functions called such as init_platform(). We will use step-over or resume. If we resume after you have provided the input of matrix 1 elements, the execution will stop at the breakpoint we have set, and we can see the updated values of Matrix 1 elements.



```
System Debugger using Debug_Matrix_Multiplication.elf on Zybo_RS (Zybo_RS)
  APU
    ARM Cortex-A9 MPCore #0 (Breakpoint: Mat_Mul.c:89)
      0x00110568 main(): ../src/Mat_Mul.c, line 89
      0x001007a8 _start(): xil-crt0.S, line 138
    ARM Cortex-A9 MPCore #1 (Suspended)
  xc7z010
```

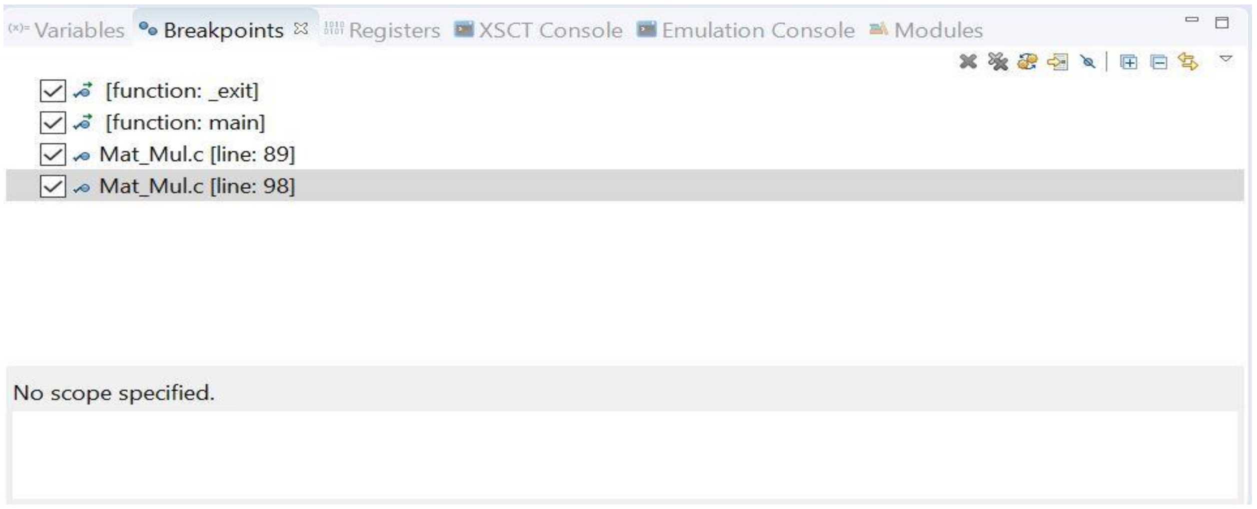
Name	Type	Value
M1	int [3][3]	[[1, 1, 1], [2, 2, 2], [3, ..
[0]	int [3]	[1, 1, 1]
[1]	int [3]	[2, 2, 2]
[2]	int [3]	[3, 3, 3]
M2	int [3][3]	[[-1, 0, -1], [0, -1, 0], [..
res	int [3][3]	[[0, -1, 0], [-1, 0, -1], [..
i	int	3

```
    printf("Enter the elements of Matrix 2\n");
    printf("==================================================\n");
    for(i=0;i<N;i++)
        for(j=0;j<N;j++)
        {
```

```
00110560:   cmp     r6, #3
00110564:   bne     -72     ; addr=0x00110524: main + 0x00000
89            printf("Enter the elements of Matrix 2\n");
00110568:   ldr     r0, [pc, #+468]
0011056c:   add     r4, sp, #64
```

TCF Debug Virtual Terminal - ARM Cortex-A9 MPCore #1

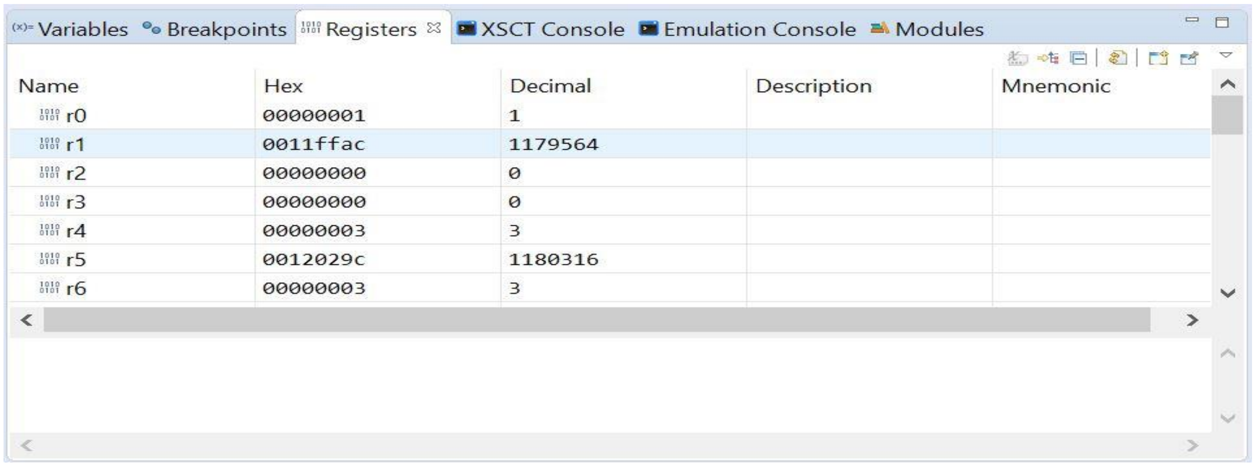
5. Breakpoints can be seen in the breakpoints view.



- [✓] [function: _exit]
- [✓] [function: main]
- [✓] Mat_Mul.c [line: 89]
- [✓] Mat_Mul.c [line: 98]

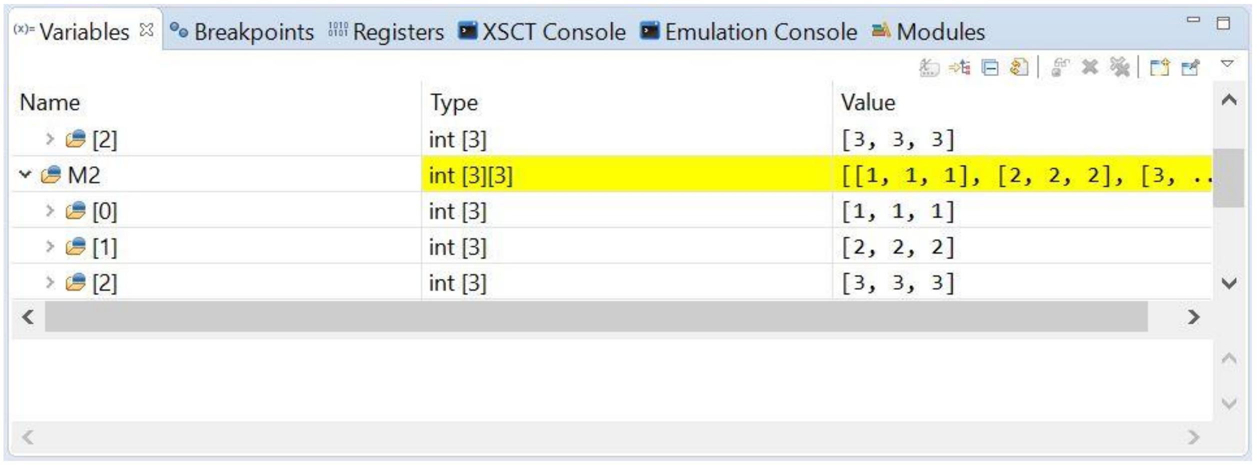
No scope specified.

6. The value of registers can be seen in the register view.

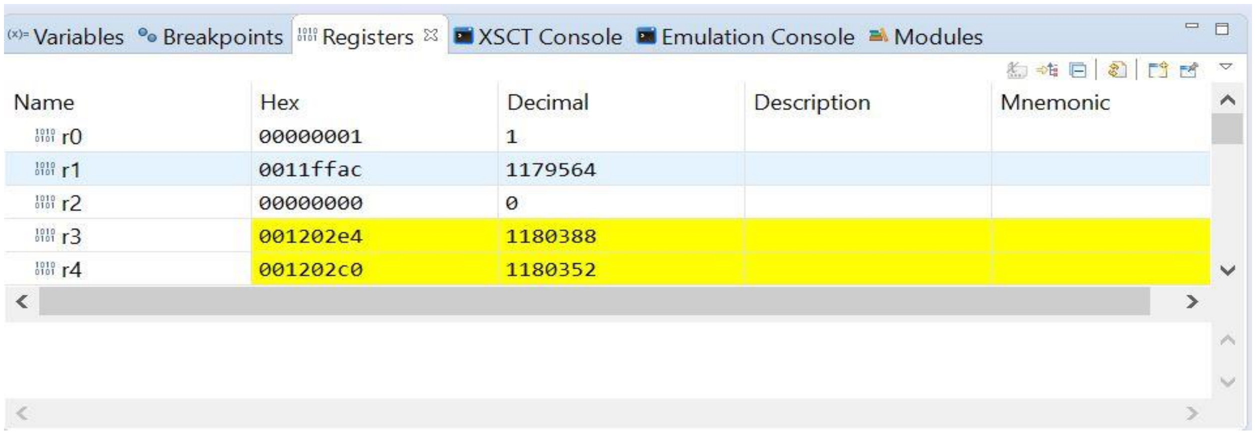


Name	Hex	Decimal	Description	Mnemonic
r0	00000001	1		
r1	0011ffac	1179564		
r2	00000000	0		
r3	00000000	0		
r4	00000003	3		
r5	0012029c	1180316		
r6	00000003	3		

7. The second breakpoint is set after the user has provided the Matrix 2 inputs. The corresponding updated variables and registers are shown below.



Name	Type	Value
> [2]	int [3]	[3, 3, 3]
⌄ M2	int [3][3]	[[1, 1, 1], [2, 2, 2], [3, ..
> [0]	int [3]	[1, 1, 1]
> [1]	int [3]	[2, 2, 2]
> [2]	int [3]	[3, 3, 3]



Name	Hex	Decimal	Description	Mnemonic
r0	00000001	1		
r1	0011ffac	1179564		
r2	00000000	0		
r3	001202e4	1180388		
r4	001202c0	1180352		

8. Add other breakpoints and explore the debugger mode more.

Submission(Graded):

1. **Paste the screenshots of Block Diagram, C Codes with comments, and JTAG Terminal Output(Part 2 and Part 3) in a PDF and submit it to the classroom.**

Self-Study Questions (Ungraded):

1. **Explore other matrix operations such as matrix addition, subtraction, transpose, etc.**