Siemens EDA Academy of Excellence

FPGA Course

Final Project

Design and implementation of Matrix Multiplier on FPGA

Supervised by: Dr / Ihab Adly

Team Names *(Group 4)*
Mohamed Sayed Mohamed
Abdullah Rajab
Ahmed Yasser

Date: 4/10/2021

Our project is about designing matrix multiplier using three different methods:

- Using FSMD (Finite State Machine with Data path)
- On NIOS processor
- On Cortex-M0 Processor

Hardware Tools we used: *FPGA DE0-Nano kit.*

Software Tools we used: *Visual Studio Code, Modelsim , Quartus 18.1,Keil uVision5*

First: Matrix Multiplication using FSMD

First we needed to think on how the matrix multiplier works, so we made a flow chart for it, and then we translated this flow chart to C code. Then we identified all the states we needed to build a design for this code, and started to write all outputs from each state and state transitions, and last we started to build the design and put all the blocks we needed and write the code.

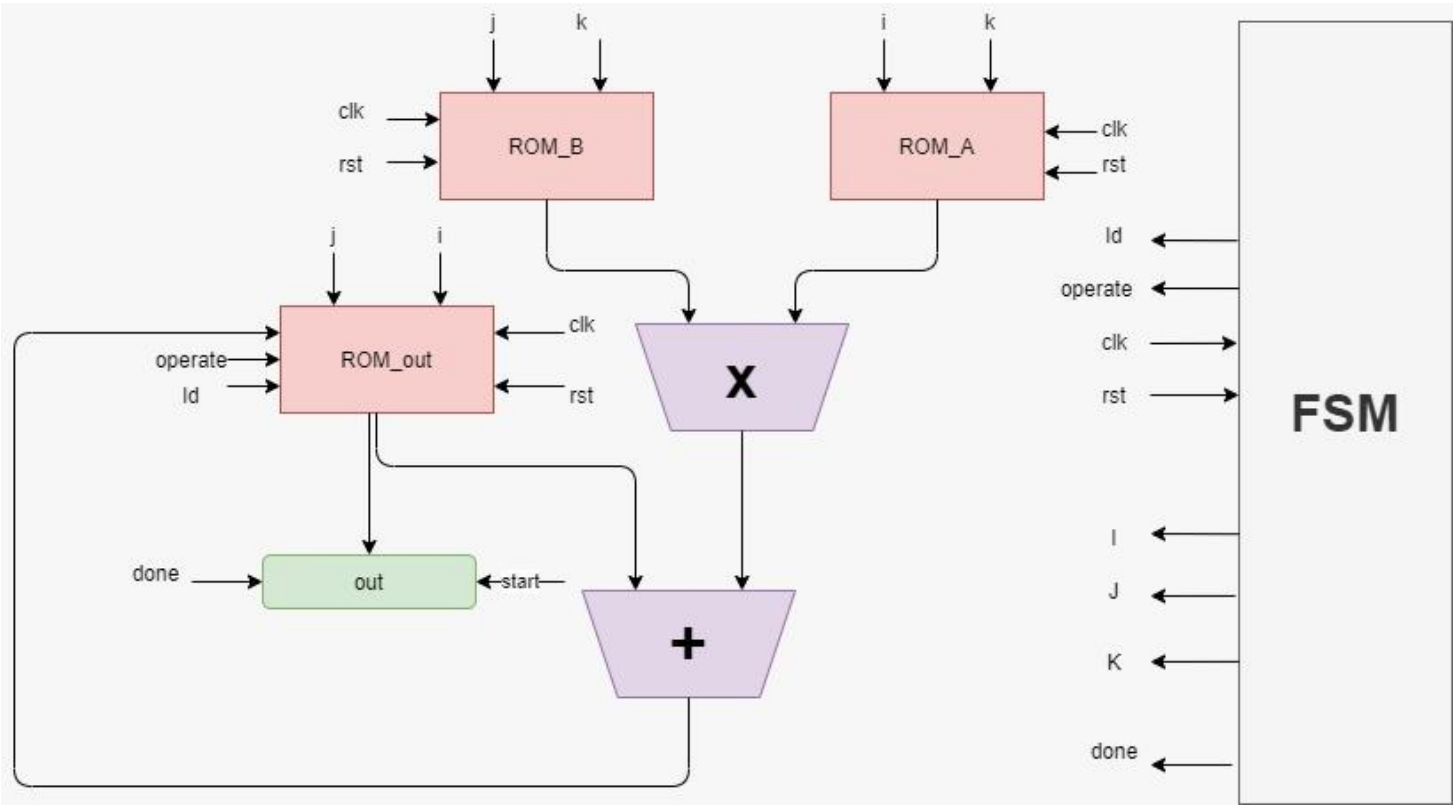
pseudo code: *(code that we made our ideas from)*

```
for(i=0;i<r;i++)
{
    for(j=0;j<c;j++)
    {
        mul[i][j]=0;
        for(k=0;k<c;k++)
        {
        mul[i][j]+=a[i][k]*b[k][j];
        }
    }
}
```

We have three loops to loop on all elements in every row and column in the two input matrices then started to multiply, add and store the final result in another matrix.

We made two designs to perform the required function correctly.

Design 1: *(based on the pseudo code)*



For <u>*Data Path*</u> we have 6 main blocks:

- ROM_A,ROM_B to store the input matrices.
- Multiplier and Adder to perform the needed multiplications and additions.
- ROM_out,Reg_out to store the results in another matrix and display it on lads on FPGA.

For <u>*Controller*</u>: (FSM)

It controls the data path by sending the signals and ensuring that the states are performed at the required time.

- I,J,K : to indexing the 3 matrices (addresses signals).
- ld : to allow ROM_out to store the result.

- operate: to allow ROM_out to output the result stored to add it with the multiplication result.
- done: output signals to inform the user that the output multiplication is ready.

And there's "start" signal to allow the output to be displayed on the leds. (We can connect it to a push button or a switch).

From DE0-Nano User Manual :

We will see that we have two buttons and both are active low,so we must put that in consideration when we put the "reset" and "start".

We have 8 leds only so we can't show a number with size greater than 7 bit if we put a led for "done" signal, or a size greater than 8 bit if we don't output the "done" signal.

Clock frequency is 50MHz so we can't show the output with that frequency and we will need a delay or a clock divider to make the outputs to be observed.

RTL CODE: (USING VERILOG)

First:Datapath code

We stored the input data in a .mem file to make it easy to change it without opening the RTL modules.

ROM_A.v

```
module ROM_A(clk,rst,i,k,out);
input clk ,rst;
input [1:0] i,k;
output reg [2:0] out;
reg[2:0] ram[0:8];
always@(posedge clk)
    begin
        if (!rst)
            begin
                out<=0;
            end
        case({i,k})
            4'b0000:    out<=ram[0];
            4'b0001:    out<=ram[1];
```

```
module ROM_B(clk,rst,j,k,out);
input clk ,rst;
input [1:0] j,k;
output reg [2:0] out;
reg[2:0] ram[0:8];
always@(posedge clk)
    begin
        if (!rst)
            begin
                out<=0;
            end
                case({k,j})
            4'b0000:    out<=ram[0];
            4'b0001:    out<=ram[1];
            4'b0010:    out<=ram[2];
```

We made a ROM for matrix A and addressed by inputs i,k, and another ROM for matrix B and addressed by inputs k,j, and concatenate the 2 inputs i,k and k,j to perform the required row and column.

We assumed that the input is 3 bit so the maximum size of output can be 8 bit.

```
module MUL(in1,in2,out);
input [2:0] in1,in2;
output [5:0] out;
```

Then the multiplier is 3 bit input, 6 bit output.

```
module adder(in1,in2,out);

input [5:0] in1;
input [6:0] in2;
output  [7:0] out;
```

The output of the multiplier is the first input of the adder, the output from ROM_out can reach 7 bit so we modified the size and the output of the adder can reach to 8 bit (maximum width).

```
MEM_OUT.v
module MEM_OUT (clk,rst ,j,i,in_adder,operate,ld,out_adder,final_out,done,start);
input clk ,rst,operate,ld,done,start;
input [1:0] j,i;
input [6:0] in_adder;
output reg [7:0] out_adder,final_out;
wire flag;
reg[7:0] ram[0:8];
reg [24:0] counter=0;
reg [3:0] cntr=0;
```

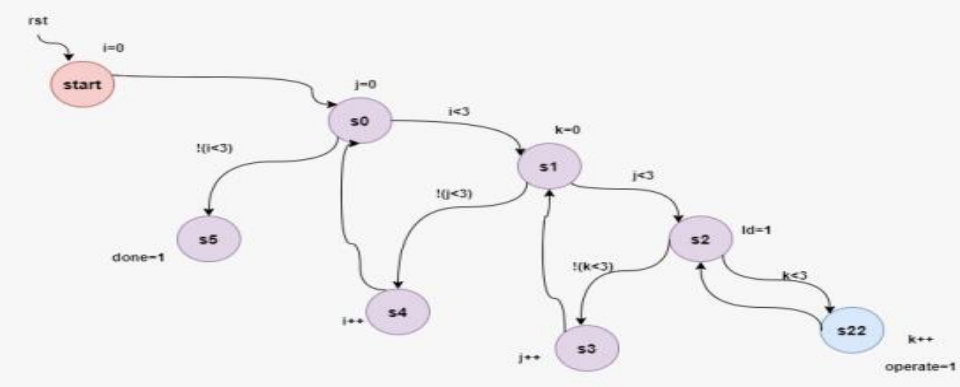
<u>MEM_OUT:</u> it's a register file with size of 9 registers with width 8 bit,and we modify it to work as a matrix by indexing it with two input signals i,j to work as the number of rows and columns.

And input signals from FSM "operate","ld","done" to operate addition, store the result and display it.

The problem we faced is that we work with frequency 50MHz so we can't notice the output, so we made a simple clock divider (counter) to slow down the result, with a "start" signal from a pushbutton to go to the next element in the result.

<u>Second : CONTROLLER CODE</u>

```
FSM.v
module FSM(clk,rst,i,j,k,operate,ld,done);
input clk,rst;
output reg done,operate,ld;
output reg [1:0] i,j,k;


localparam s0=3'b000, s1=3'b001, s2=3'b010,s22=3'b111, s3=3'b011 , s4=3'b100, s5=3'b101 , start_state=3'b110;
reg [2:0] present_state,next_state;
always @(posedge clk)
    begin
```



We have 8 states to operate the function correctly, we can reduce them to make the result be ready faster but we work on high frequency so it's okay.

Pin Planner

Named: * Edit:

Node Name	Direction	Location	I/O Bank	VREF Group	Fitter Location	I/O Standard
clk	Input	PIN_R8	3	B3_N0	PIN_E1	2.5 V (default)
done	Output	PIN_L3	2	B2_N0	PIN_A15	2.5 V (default)
final_out[7]	Output	PIN_D5	8	B8_N0	PIN_N6	2.5 V (default)
final_out[6]	Output	PIN_B1	1	B1_N0	PIN_D8	2.5 V (default)
final_out[5]	Output	PIN_F3	1	B1_N0	PIN_B11	2.5 V (default)
final_out[4]	Output	PIN_D1	1	B1_N0	PIN_C11	2.5 V (default)
final_out[3]	Output	PIN_A11	7	B7_N0	PIN_C9	2.5 V (default)
final_out[2]	Output	PIN_B13	7	B7_N0	PIN_E9	2.5 V (default)
final_out[1]	Output	PIN_A13	7	B7_N0	PIN_A10	2.5 V (default)
final_out[0]	Output	PIN_A15	7	B7_N0	PIN_B10	2.5 V (default)
rst	Input	PIN_E1	1	B1_N0	PIN_F9	2.5 V (default)
start	Input	PIN_J15	5	B5_N0	PIN_D9	2.5 V (default)
<<new node>>						

We connected the output result to 7 leds and the last led is for the “done” signal , start and reset to the active low push buttons, clk to the internal clock of the kit.

Resources Results

Entity:Instance

Cyclone IV E: EP4CE22F17C6

matrix_mul

Tasks Compilation

Task
Compile Design
Analysis & Synthesis
Fitter (Place & Route)
Assembler (Generate programm

Table of Contents

- Flow Summary
- Flow Settings
- Flow Non-Default Global Sett
- Flow Elapsed Time
- Flow OS Summary
- Flow Log
- Analysis & Synthesis
- Fitter
- Assembler
- Timing Analyzer
- EDA Netlist Writer
- Flow Messages
- Flow Suppressed Messages

Flow Summary

<<Filter>>

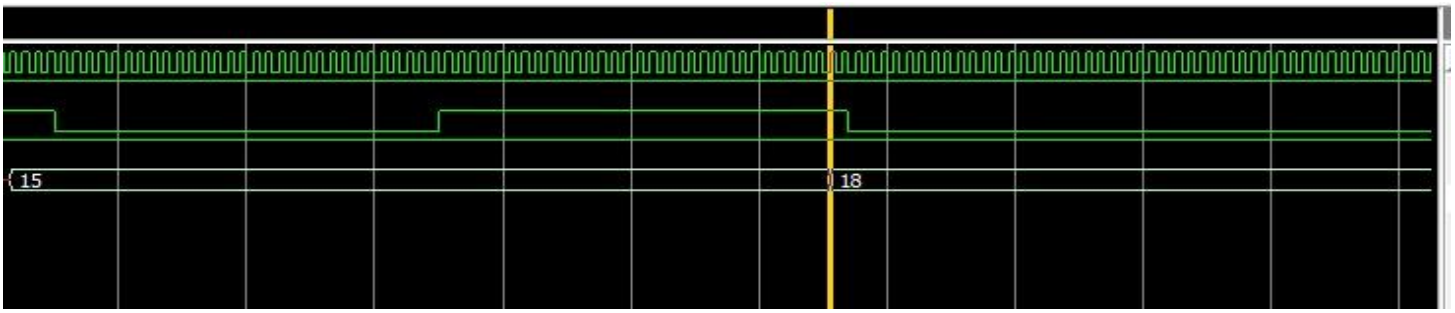
Flow Status	Successful - Mon Oct 04 12:00:11 2021
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	matrix_mul
Top-level Entity Name	matrix_mul
Family	Cyclone IV E
Device	EP4CE22F17C6
Timing Models	Final
Total logic elements	96 / 22,320 (< 1 %)
Total registers	62
Total pins	12 / 154 (8 %)
Total virtual pins	0
Total memory bits	0 / 608,256 (0 %)
Embedded Multiplier 9-bit elements	0 / 132 (0 %)
Total PLLs	0 / 4 (0 %)

All <<Filter>> Find... Find Next

```
Type   ID     Message
       204019 Generated file matrix_mul_6_1200mv_0c_vhd_slow.sdo in folder "C:/Users/User/Desktop/final_final/quartus/mat
       204019 Generated file matrix_mul_min_1200mv_0c_vhd_fast.sdo in folder "C:/Users/User/Desktop/final_final/quartus/m
       204019 Generated file matrix_mul_vhd.sdo in folder "C:/Users/User/Desktop/final_final/quartus/matrix_mul/simulatic
              Quartus Prime EDA Netlist Writer was successful. 0 errors, 0 warnings
       293000 Quartus Prime Full Compilation was successful. 0 errors, 10 warnings
```

Simulation Results

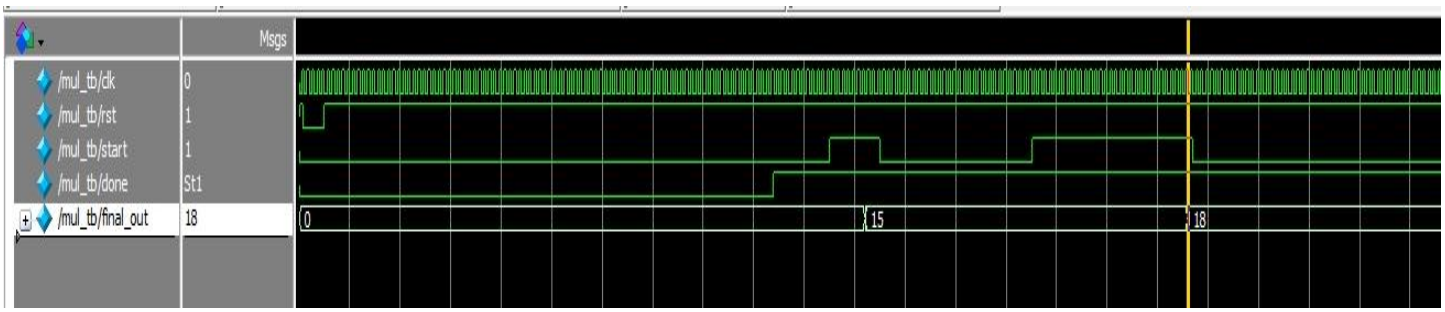
Functional Simulation:



The result of the multiplication is correct and the next result is shown only when the start signal is activated.

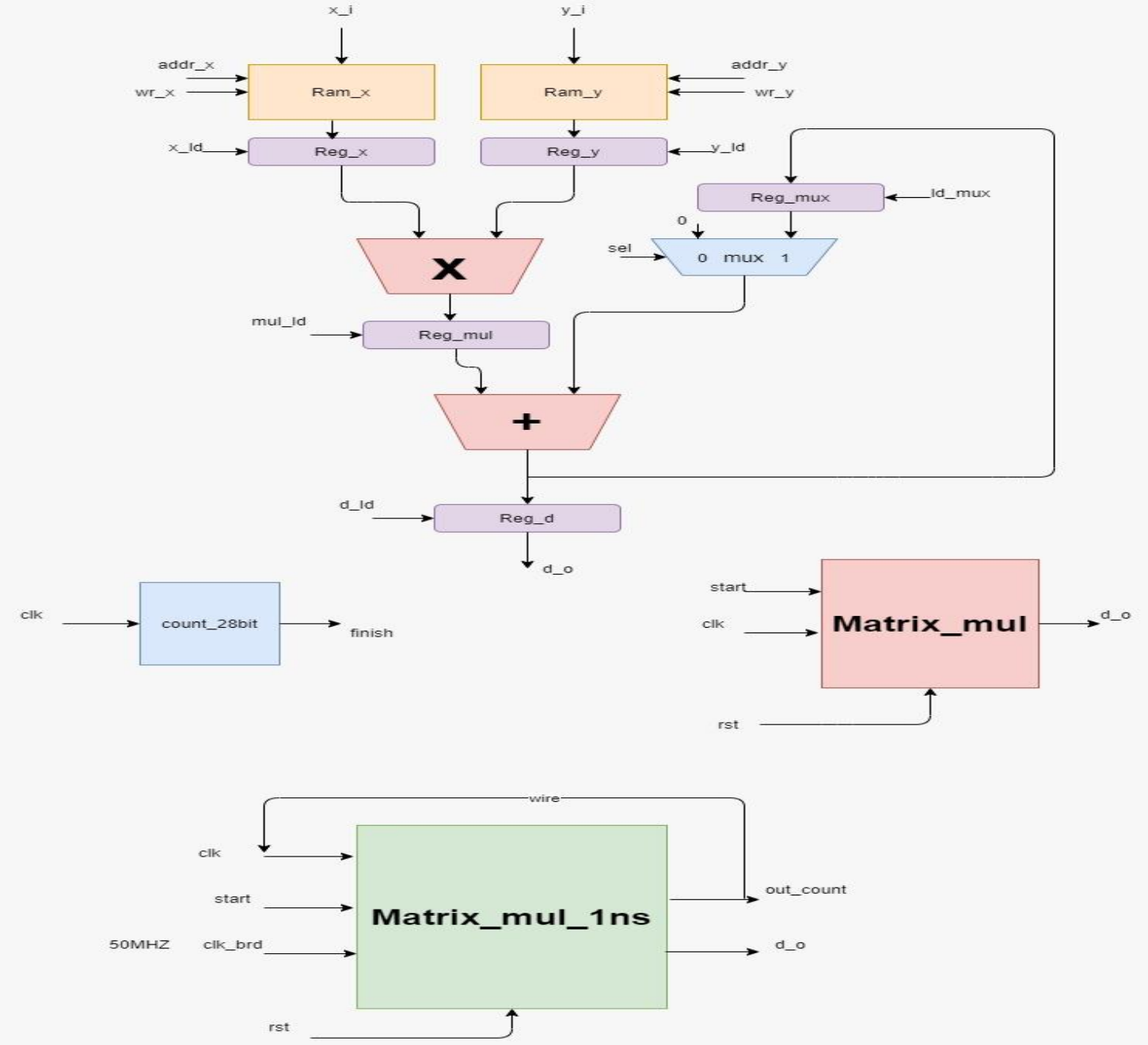
It took a period of 1820 ns to make the nine elements be ready with clock period =20ns , that means it took 91 clock cycles, that's why we didn't put a clock divider for the global clock and slow the whole system as it will take much time to display the first element.

Time Simulation:



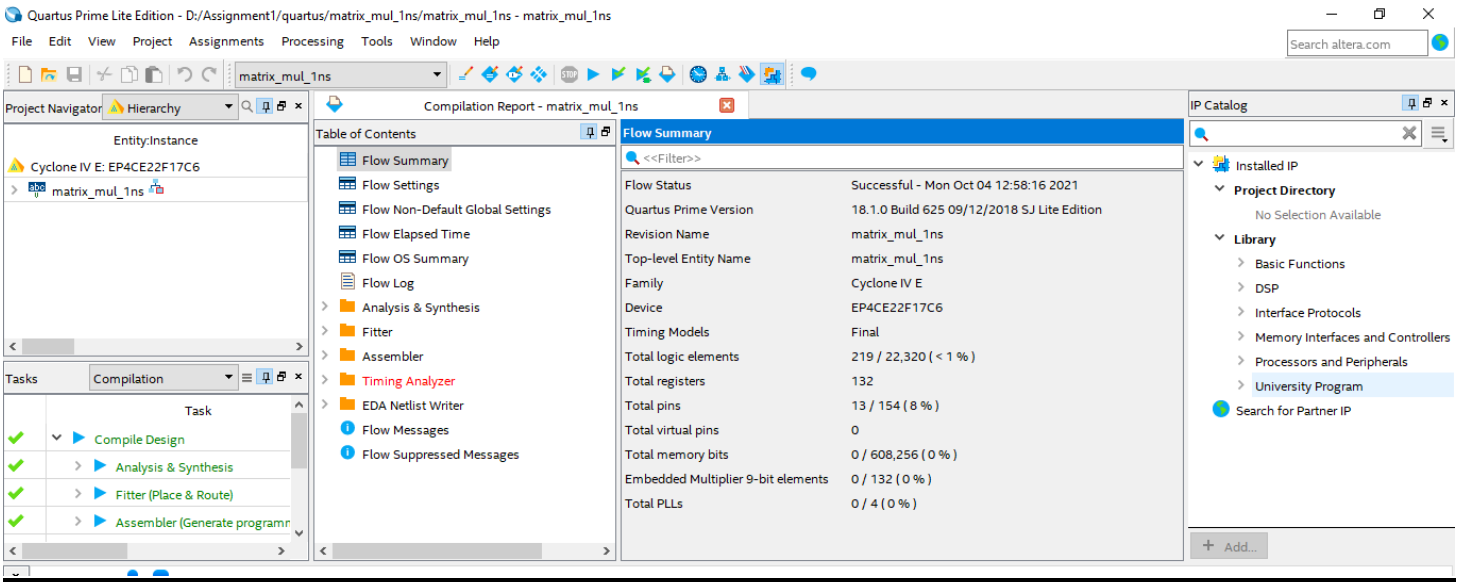
Time simulation works fine also, and the first output appears after 1820 ns (91 clock cycles).

Design 2: *(based on pipelining)*

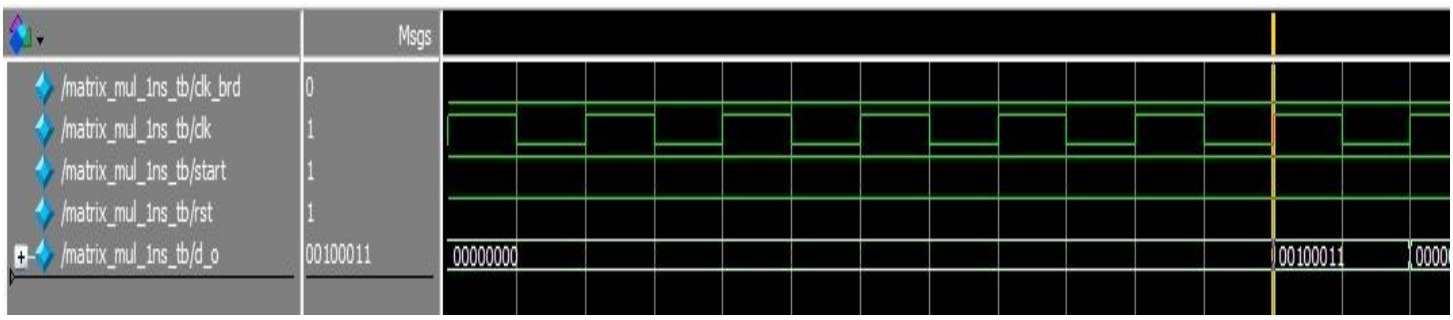


This design is optimization of the first design, it allows multiple elements to be executed in the same time using pipelining, i.e. when first element is in add state the second element will be on multiplication state and so on.

Resources Results



Simulation Results



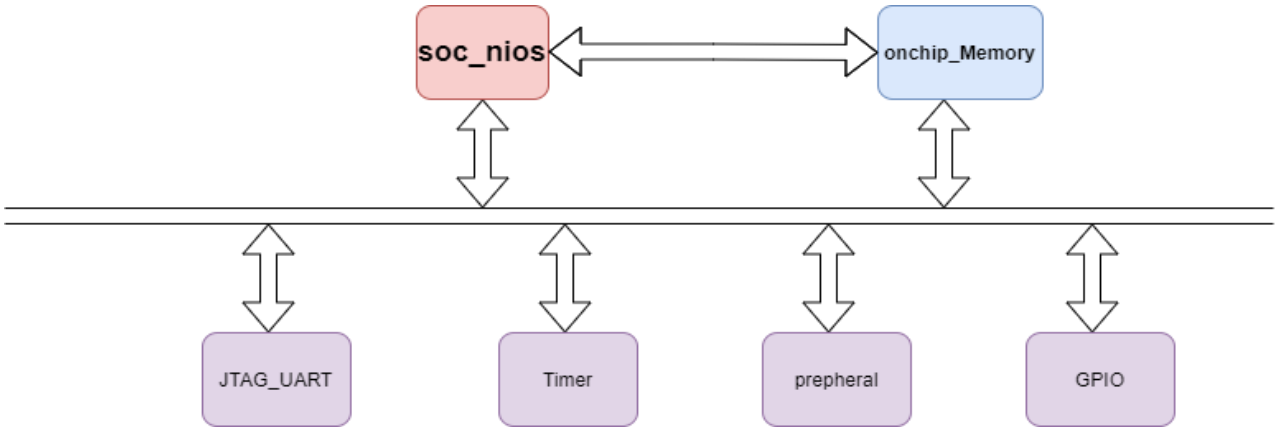
Here first element is shown after 165ns only with clock period 20ns, that means first element is shown after 8 clock cycles only, and that's faster than the first design but it's longer and complicated so we can have many issues inside and it will be difficult to trace it.

All we need now is to test the two designs on FPGA

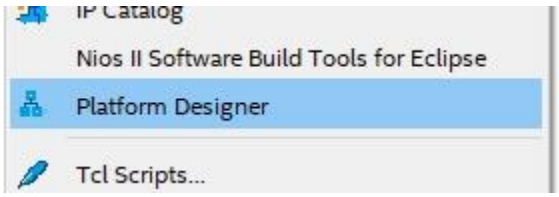
Note: We tested the two designs on FPGA and ***BOTH WORKED!***

Matrix Multiplication on NIOS processor

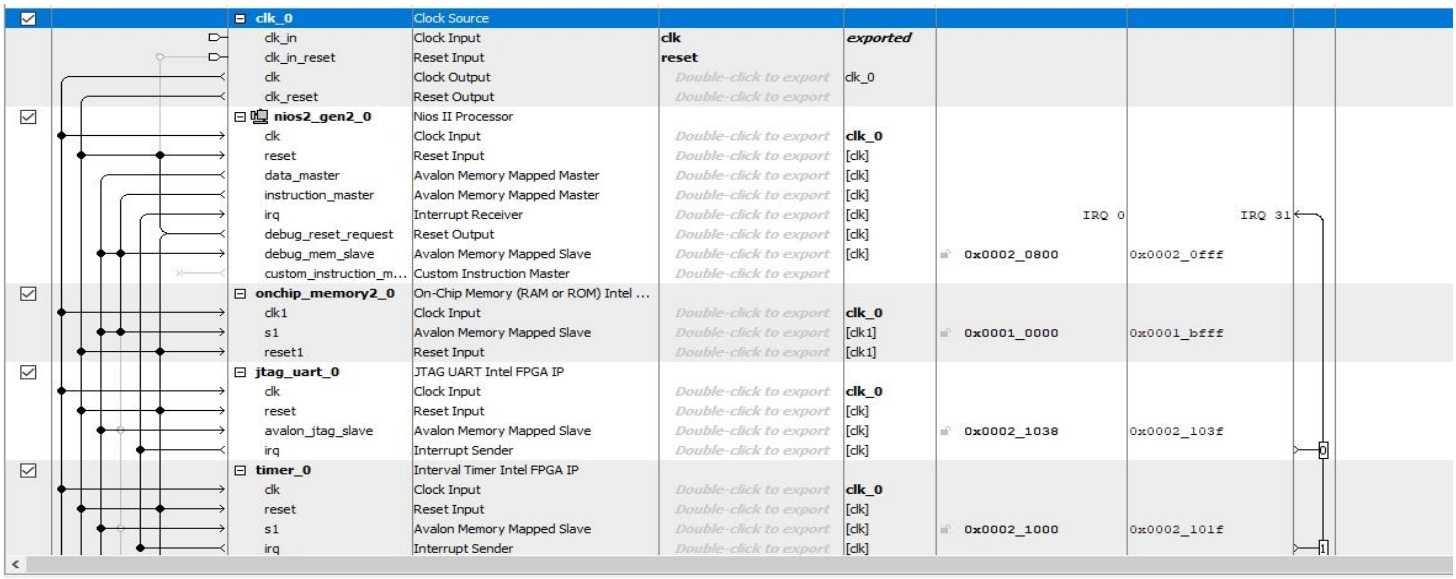
“SOC-NIOS” processor architecture:



First to design nios using quartus : tools/platform designer



Then started configuration: for memory we created data memory and instruction memory (RAM, ROM), JTAG, TIMER, GPIO, system-id, global clock and global reset and connected them together as shown in figure.



We created the memory and the main peripherals (JTAG-UART, TIMER, GPIO).

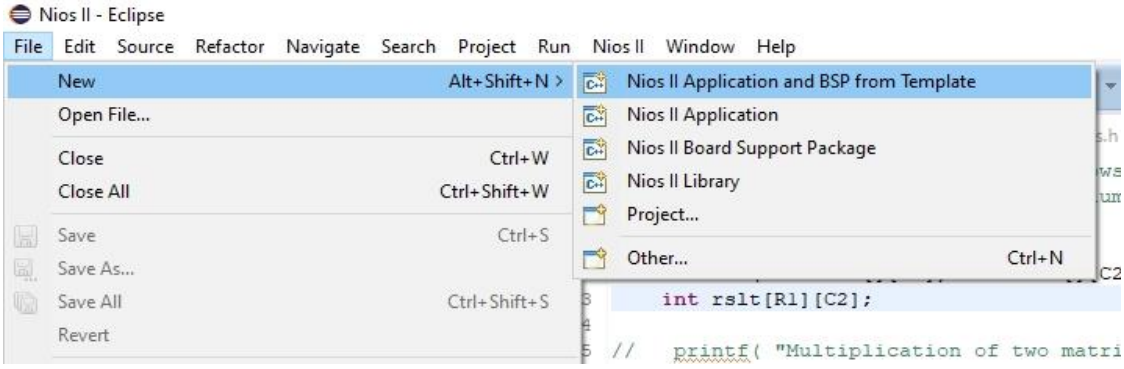
Resources Results:

Flow Status	Successful - Mon Oct 04 11:09:28 2021
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	soc_nios_fpga
Top-level Entity Name	soc_nios_fpga
Family	Cyclone IV E
Device	EP4CE22F17C6
Timing Models	Final
Total logic elements	1,855 / 22,320 (8 %)
Total registers	1039
Total pins	10 / 154 (6 %)
Total virtual pins	0
Total memory bits	404,480 / 608,256 (66 %)
Embedded Multiplier 9-bit elements	0 / 132 (0 %)
Total PLLs	0 / 4 (0 %)

Then we move on to ***Nios || software build tool for Eclipse*** to build c code of matrix multiplication.

From Eclipse :

- Create new file:

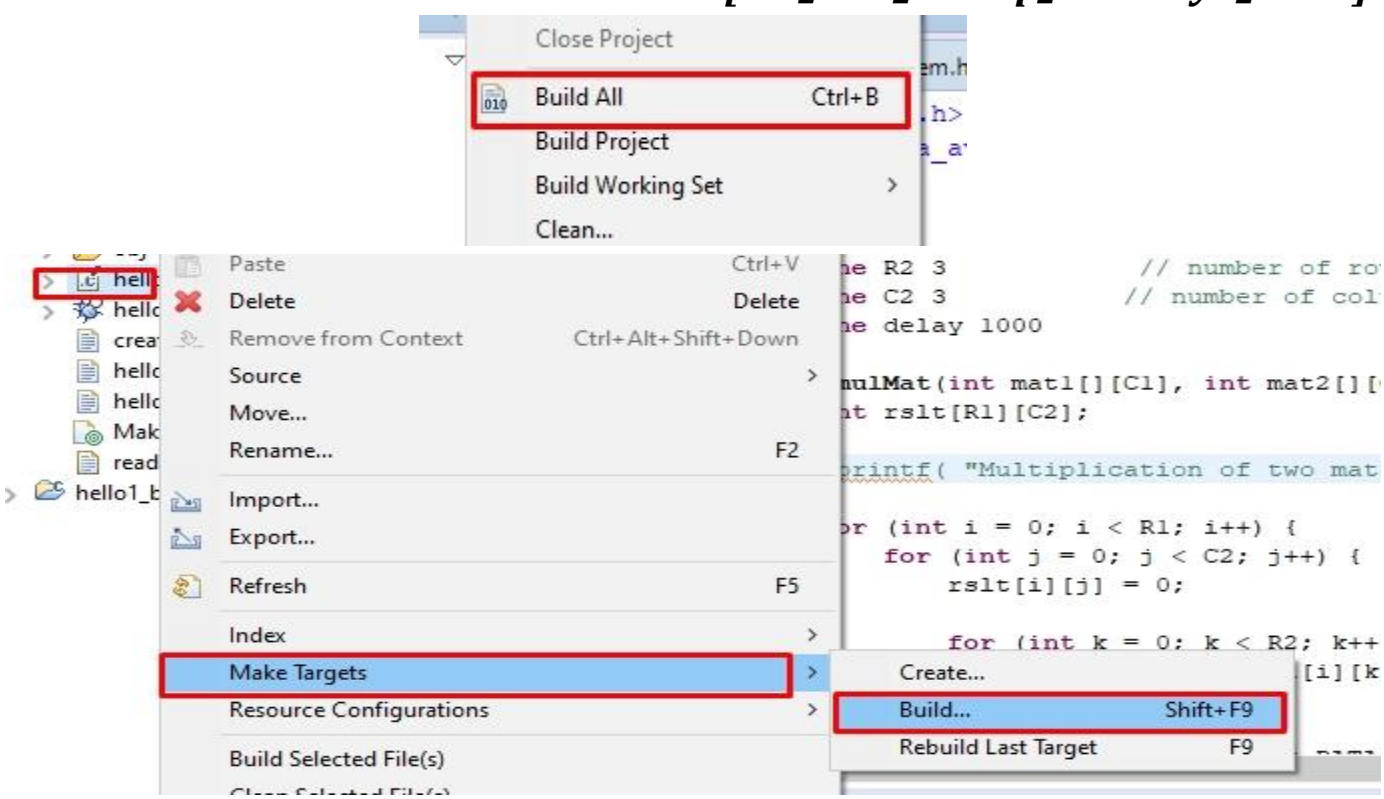


- Write C code:

By using **PIO** peripheral **[IOWR_ALTERA_AVALON_PIO_DATA(base, data)]** with address **[PIO_0_BASE 0x21020].**

```c
#define R1 3                // number of rows in Matrix-1
#define C1 3                // number of columns in Matrix-1
#define R2 3                // number of rows in Matrix-2
#define C2 3               // number of columns in Matrix-2
#define delay 2000000

void mulMat(int mat1[][C1], int mat2[][C2]) {
    int rslt[R1][C2];

//   printf( "Multiplication of two matrices is:\n" );

    for (int i = 0; i < R1; i++) {
        for (int j = 0; j < C2; j++) {
            rslt[i][j] = 0;

            for (int k = 0; k < R2; k++) {
                rslt[i][j] += mat1[i][k] * mat2[k][j];
            }

            IOWR_ALTERA_AVALON_PIO_DATA(0x21020, rslt[i][j]);
            for (int d=0; d<delay; d++) ;
        }

    }
```

- Build the c code then create .hex file ***[soc_nios_onchip_memory2_0.hex]:***



Then copy it to soc_nios simulation then we create **soc_nios.vhd.**

```
entity soc_nios_fpga is
  port (clk,rst: in std_logic;
        leds: out std_logic_vector(7 downto 0));
 end soc_nios_fpga ;

architecture struct of soc_nios_fpga is
    component soc_nios is
        port (
            clk_clk                          : in  std_logic                     := 'X'; -- clk
            reset_reset_n                    : in  std_logic                     := 'X'; -- reset_n
            pio_0_external_connection_export : out std_logic_vector(7 downto 0)         -- export
        );
    end component soc_nios;
begin
       u0 : component soc_nios

        port map (
            clk_clk                          => clk,  --                        clk.clk
            reset_reset_n                    => rst,  --                      reset.reset_n
            pio_0_external_connection_export => leds    -- pio_0_external_connection.export
        );
```

Then by using modelsim to simulation by using this command:

Vsim -do .\msim_setup.tcl to show number of cycles.

```
PS E:\engineering\siemens\vs_code\projects\lab6> cd .\quartus\soc_nios\soc_nios\simulation\mentor\
PS E:\engineering\siemens\vs_code\projects\lab6\quartus\soc_nios\soc_nios\simulation\mentor> vsim -do .\msim_setup.tcl
```

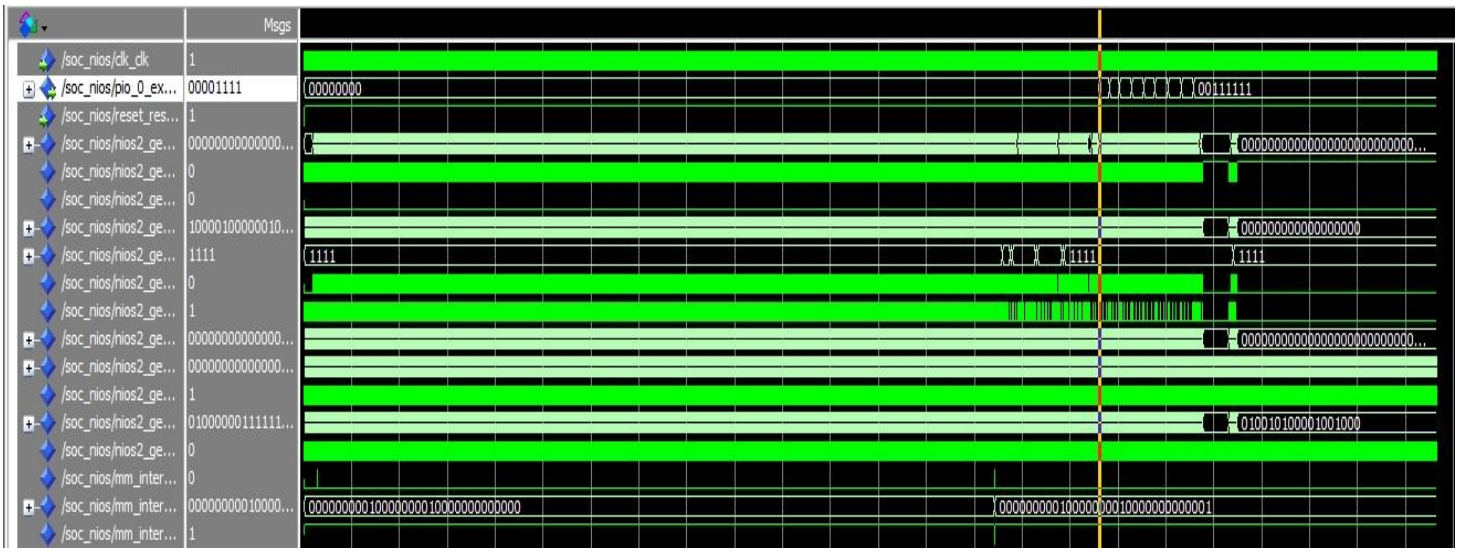
We write three commands on the transcript: dev_com , com , elab

Then define clock and reset:

```
force -freeze sim:/soc_nios/clk_clk 1 0, 0 {10000 ps} -r 20000
force -freeze sim:/soc_nios/reset_reset_n 1 0
force -freeze sim:/soc_nios/reset_reset_n 0 5000
force -freeze sim:/soc_nios/reset_reset_n 1 10000
```

Than start simulation for 30ns.

Simulation Results



First output is shown after 3.32242ms that means 166121 clock cycles, and it took a period of 42.04 us that mean 2102 clock cycles.

The 9 numbers will be displayed in a period of 294.25 us (14714 clock cycles).

So we need to add delay to make it easy to observe.
We added a loop counts to 1000 and it added 1ms delay so if we added a loop to count to 1000000 it will add a delay of 1 sec, and then it will be easy to be observed on FPGA.

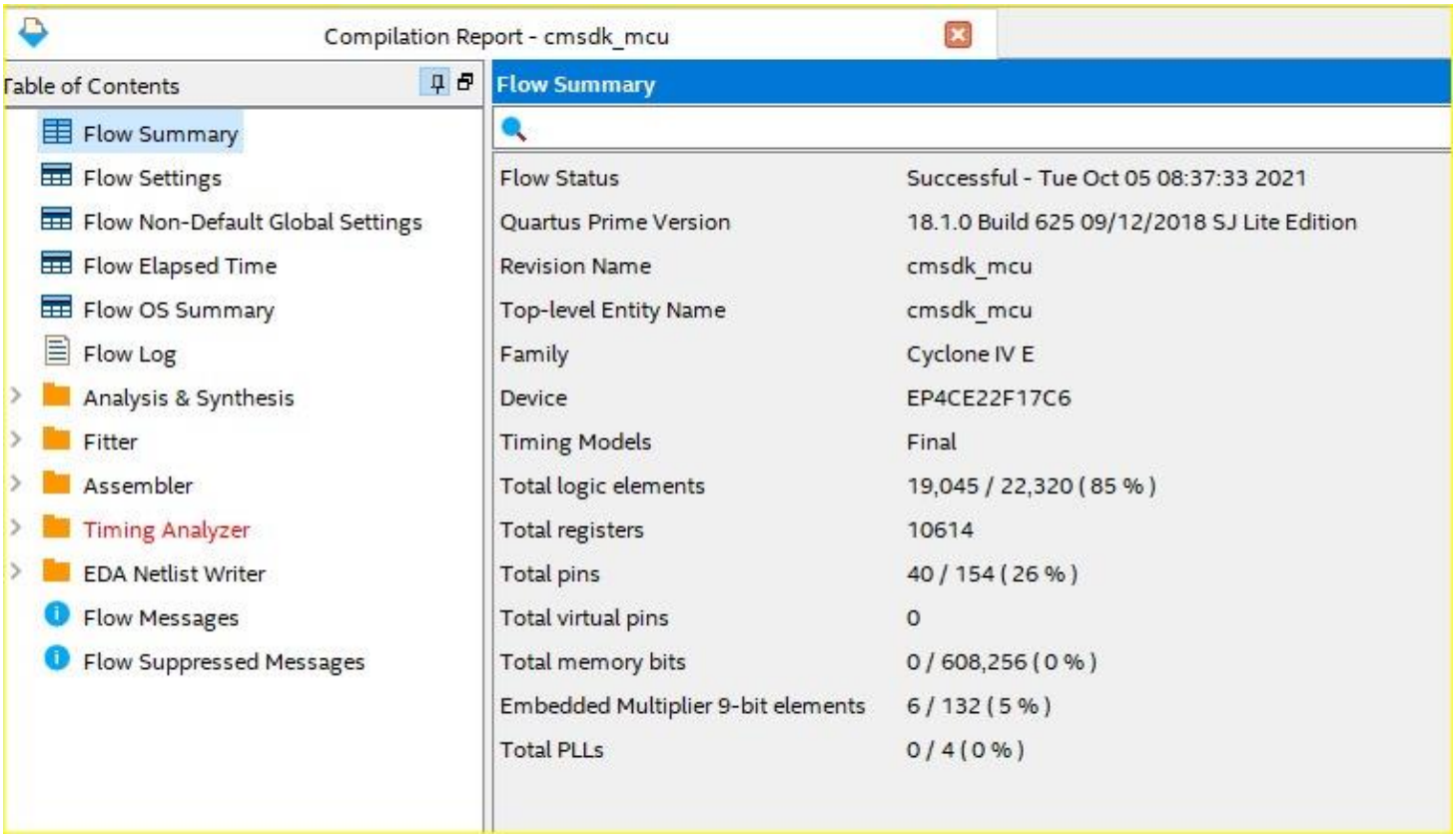
Matrix Multiplication on Cortex-M0 processor

Cortex_m0 simulation:

- Description:
  To multiply two matrices of size 3 by 3 and then print the answer.

- Tools:
  1. We use KEIL micro vision 5 to implement the HDL language of the Cortex_mo processor that takes the hex version of our software code.
  2. We use modelsim altera to simulate the RTL design from KEIL Tool.

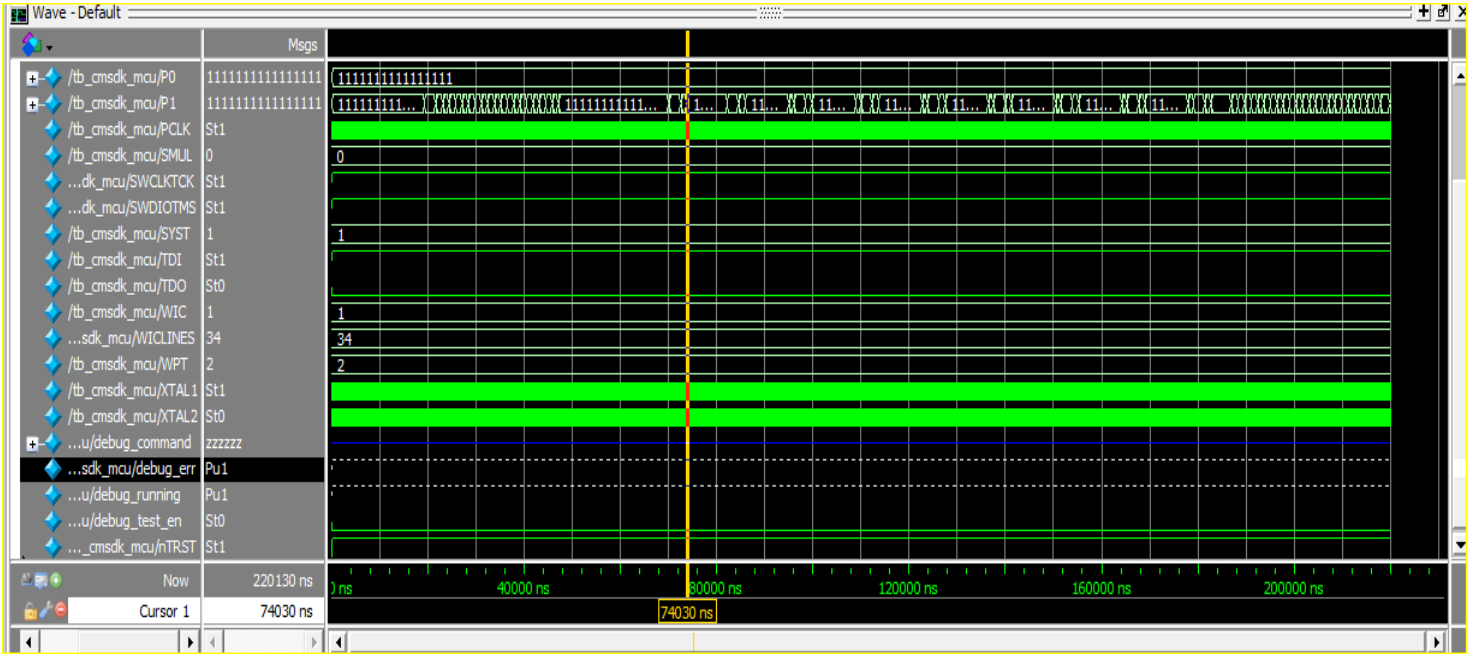
- Resources:



- Code:

```c
#ifdef CORTEX_M0
#include "CMSDK_CM0.h"
#include "core_cm0.h"
#endif
#ifdef CORTEX_M0PLUS
#include "CMSDK_CM0plus.h"
#include "core_cm0plus.h"
#endif
#include <stdio.h>
#include "uart_stdout.h"
int main (void)
{
    int x[3][3] ={{3,0,1},{2,3,4},{5,7,1}};
    int y[3][3] ={{1,3,2},{4,7,2},{1,0,6}};
    int z[3][3], t[3];
    int i,j,k;
    CMSDK_GPIO0->OUTENABLESET = 0xffff;
    for(j=0;j<=2;j++)
    {
        for(k=0;k<=2;k++)
        {
            for(i=0;i<=2;i++)
            {
             t[i] = x[j][i] * y[i][k];
        }
        z[j][k] = t[0] +t[1] +t[2];
      }
    }
    for(i=0; i<=2; i++)
    {
   for(j=0;j<=2;j++)
        {
            CMSDK_GPIO0->DATA = z[i][j];
            for(k=0;k<=100000000;k++) {}
    }
```

- simulation:



```
Transcript
VSIM 2> run -all
# 47370 ns UART: start of simulatio
# 74030 ns UART: 4
# 86110 ns UART: 9
# 99890 ns UART: 12
# 113830 ns UART: 18
# 127610 ns UART: 27
# 141390 ns UART: 34
# 155330 ns UART: 34
# 169110 ns UART: 64
# 182890 ns UART: 30
# 217950 ns UART: ** end of simulat
# 220130 ns UART: Test Ended
```

From simulation

The processing time from start until displaying the first output is 74030 ns (where the clock period is 20 ns) so it takes about 3701 clock cycle.