Juan Arturo Garza

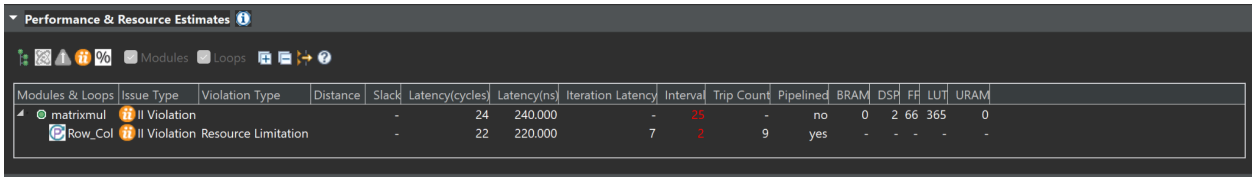
September 29th 2022

Elec 522

High Level Design

The purpose of this project is to multiply two matrices systolically; however instead of using simulink and laying out the hardware blocks, it is done in HLS in C++. The challenges I faced were trade offs in design and in the timing that came up.

Base layout (pre optimization):



Modules & Loops	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
matrixmul	II Violation			-	24	240.000	-	25	-	no	0	2	66	365	0
Row_Col	II Violation	Resource Limitation		-	22	220.000	7	2	9	yes	-	-	-	-	-

This code is the baseline which I will compare my solution to.

Final Design Choices

I plan to use the following

- HLS Parameters: HLS Array_Reshape, with dim=n
  - This will allow for faster read times of the arrays
- HLS Interface mode= ap_fifo
  - This will allow inputs to be fed systolic way when it is in similink
- HLS Pipeline
  - This will reduce the clock cycles however can cause timing violations when reading and writing

Design 1:

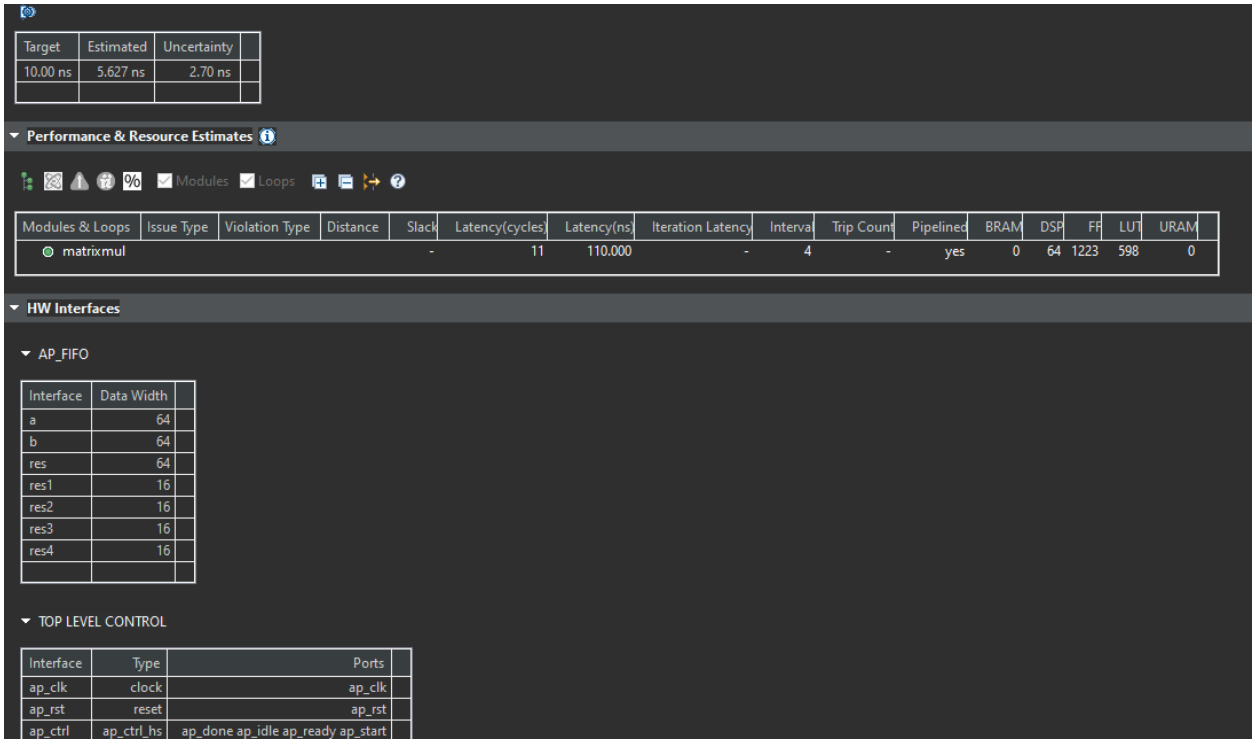
- HLS Parameters: HLS Array_Reshape, with dim=2 for both arrays
- HLS Interface mode= ap_fifo

Design 2

- HLS Parameters: HLS Array_Reshape, with dim=3 for both arrays
- HLS Interface mode= ap_fifo
- Pipeline II=1
- Matrix output was an Array of arrays which was outputted as one

Design 3 ** FINAL DESIGN

- HLS Parameters: HLS Array_Reshape, with dim=2 for both arrays
- HLS Interface mode= ap_fifo
- Pipeline II=4
    - No Violations



Target	Estimated	Uncertainty
10.00 ns	5.627 ns	2.70 ns

Performance & Resource Estimates

Modules Loops

Modules & Loops	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
matrixmul				-	11	110.000	-	4	-	yes	0	64	1223	598	0

HW Interfaces

AP_FIFO

Interface	Data Width
a	64
b	64
res	64
res1	16
res2	16
res3	16
res4	16

TOP LEVEL CONTROL

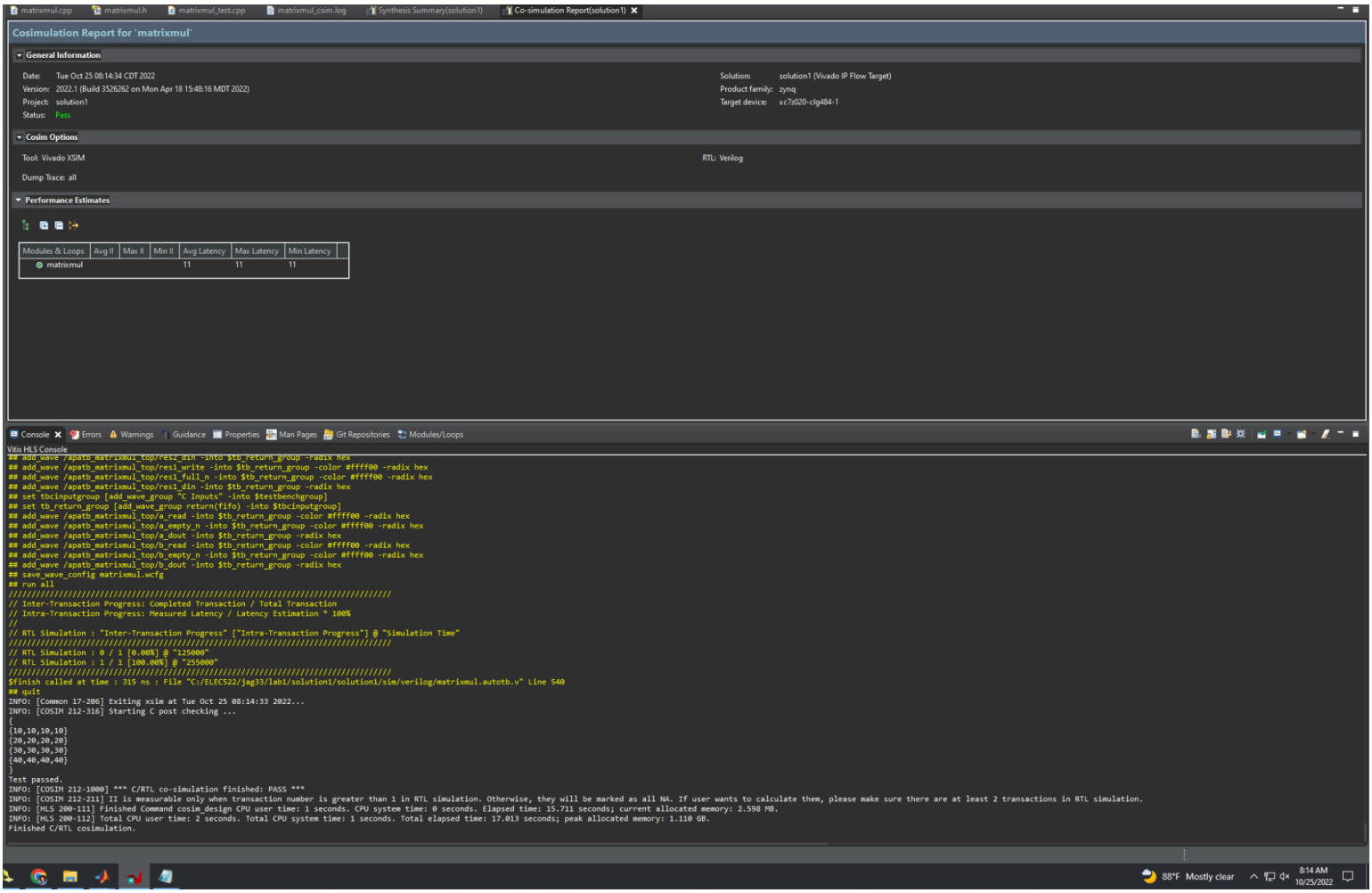
Interface	Type	Ports
ap_clk	clock	ap_clk
ap_rst	reset	ap_rst
ap_ctrl	ap_ctrl_hs	ap_done ap_idle ap_ready ap_start

This design has matrix A and Matrix B fed into as one big array and then the output fed out as several rows of the resultant matrix. My design uses Pipelining and a fifo structure. It is able to get results after 14 cycles.

I wil;l not that changing the number type in HLS for matrix A and matrix B would decrease DSP and latency however it would not work in model composer and was thus changed to be shorts. This is a drawback to my design and needs further change.

This design uses more DSP but works in Model composer and passes all test cases

Co-sim Passing:



Test Bench

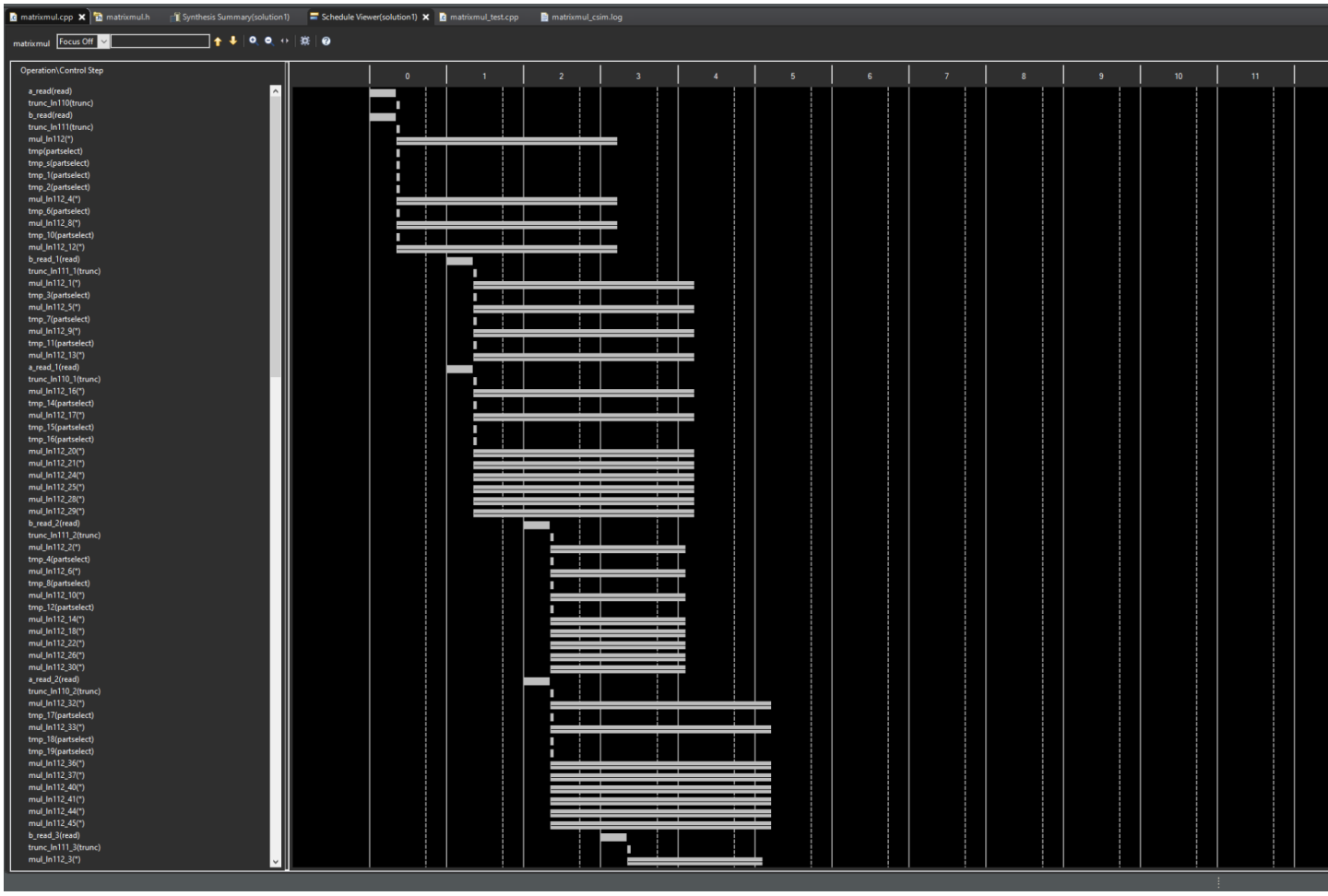
Passed with no errors from hardware and software. Both Aspects agreed on output

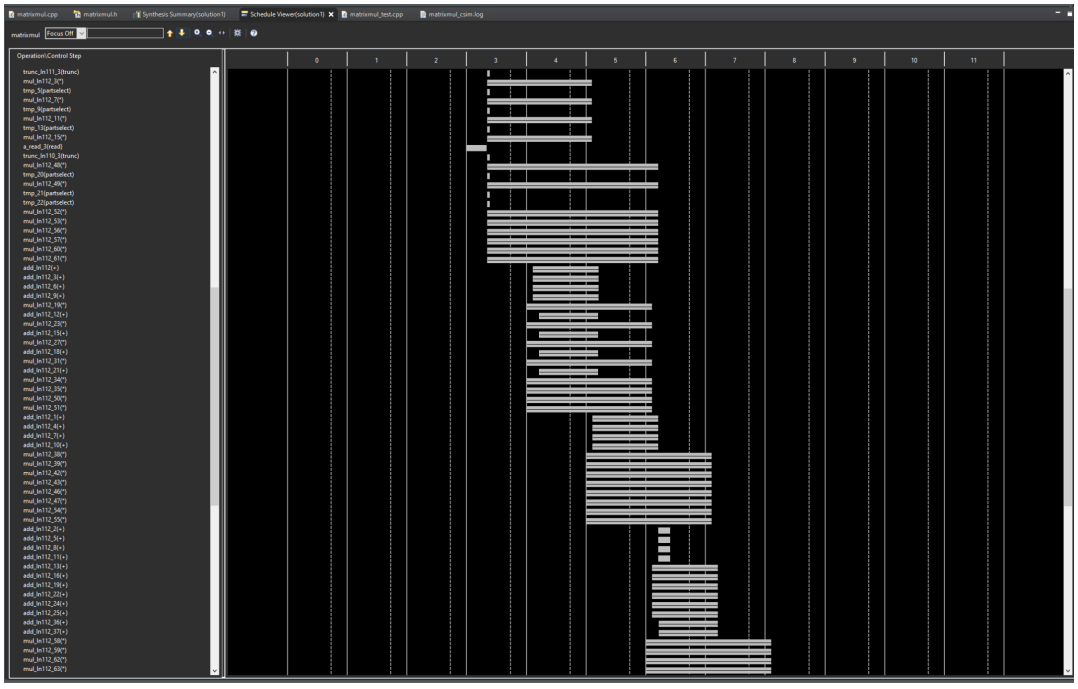
```
#include <iostream>
#include "matrixmul.h"

using namespace std;

int main(int argc, char **argv)
{
  mat_a_t in_mat_a[4][4] = {
     {1, 1, 1, 1},
     {2, 2, 2, 2},
     {3, 3, 3, 3},
     {4, 4, 4, 4}
  };
  mat_b_t in_mat_b[4][4] = {
     {1, 1, 1, 1},
     {2, 2, 2, 2},
     {3, 3, 3, 3},
     {4, 4, 4, 4}
  };
  result_t hw_result[4][4], sw_result[4][4];
  //short sw_result[4][4];
  result_t_r1 hw_result_1[4];
  result_t_r2 hw_result_2[4];
  result_t_r3 hw_result_3[4];
  result_t_r4 hw_result_4[4];
  int err_cnt = 0;

  for(int i = 0; i < MAT_A_ROWS; i++) {
     for(int j = 0; j < MAT_B_COLS; j++) {
        sw_result[i][j] = 0;

        for(int k = 0; k < MAT_B_ROWS; k++) {
           sw_result[i][j] += in_mat_a[i][k] * in_mat_b[k][j];
        }
     }
  }
```

```
INFO: [SIM 2] *************** CSIM start ***************
INFO: [SIM 4] CSIM will launch GCC as the compiler.
   Compiling ../../../../matrixmul_test.cpp in debug mode
   Generating csim.exe
{
{10,10,10,10}
{20,20,20,20}
{30,30,30,30}
{40,40,40,40}
}
Test passed.
INFO: [SIM 1] CSim done with 0 errors.
INFO: [SIM 3] *************** CSIM finish ***************

```

Note that my testbench compares the row row output which are four arrays to the software result which is one array.

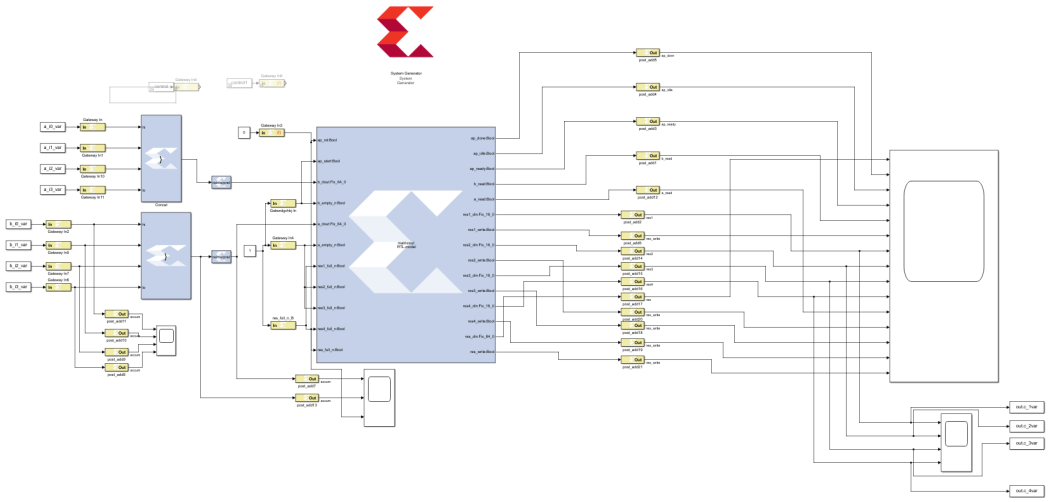
Timing:


matrixmul.cpp
matrixmul.h
Synthesis Summary(solution1)
Schedule Viewer(solution1)
matrixmul_test.cpp
matrixmul_csim.log
matrixmul
Focus Off
Operation\Control Step
0
1
2
3
4
5
6
7
8
9
10
11
a_read(read)
trunc_ln110(trunc)
b_read(read)
trunc_ln111(trunc)
mul_ln112(*)
tmp(partselect)
tmp_s(partselect)
tmp_1(partselect)
tmp_2(partselect)
mul_ln112_4(*)
tmp_6(partselect)
mul_ln112_6(*)
tmp_10(partselect)
mul_ln112_12(*)
b_read_1(read)
trunc_ln111_1(trunc)
mul_ln112_1(*)
tmp_3(partselect)
mul_ln112_5(*)
tmp_7(partselect)
mul_ln112_9(*)
tmp_11(partselect)
mul_ln112_13(*)
a_read_1(read)
trunc_ln110_1(trunc)
mul_ln112_16(*)
tmp_14(partselect)
mul_ln112_17(*)
tmp_15(partselect)
tmp_16(partselect)
mul_ln112_20(*)
mul_ln112_21(*)
mul_ln112_24(*)
mul_ln112_25(*)
mul_ln112_28(*)
mul_ln112_29(*)
b_read_2(read)
trunc_ln111_2(trunc)
mul_ln112_2(*)
tmp_4(partselect)
mul_ln112_6(*)
tmp_8(partselect)
mul_ln112_10(*)
tmp_12(partselect)
mul_ln112_14(*)
mul_ln112_18(*)
mul_ln112_22(*)
mul_ln112_26(*)
mul_ln112_30(*)
a_read_2(read)
trunc_ln110_2(trunc)
mul_ln112_32(*)
tmp_17(partselect)
mul_ln112_33(*)
tmp_18(partselect)
tmp_19(partselect)
mul_ln112_36(*)
mul_ln112_37(*)
mul_ln112_40(*)
mul_ln112_41(*)
mul_ln112_44(*)
mul_ln112_45(*)
b_read_3(read)
trunc_ln111_3(trunc)
mul_ln112_3(*)

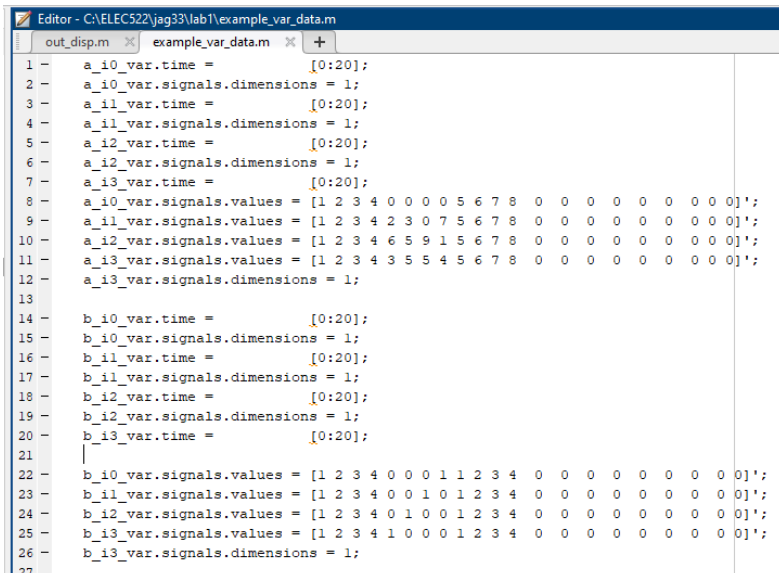


The timing sheet shows that there are multiple adds and multiples which most cycle times and overlap, meaning that it is done in parallel. The drawback of this is that it uses more resources to finish faster as more is done in parallel.

Model Composer



Inputs:



Note that there is no padding for the inputs of the matrix, every 4 clock cycles there is a new matrix, The matrix is also fed by column, not by row, thus an example: the bottom row of b_i3 is the leftmost column of the matrix, however the output is fed by row instead of column. For Model composer there are three different matrices used and tested one after the other

Output:

```
>> out_disp
   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN    10    10    10    10     3     6     2     0    26    26    26    26     0     0     0     0     0     0     0     0     0     0

   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN    20    20    20    20     5     5     3     0    52    52    52    52     0     0     0     0     0     0     0     0     0     0

   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN    30    30    30    30     5     9     0     0    78    78    78    78     0     0     0     0     0     0     0     0     0     0

   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN    40    40    40    40     4     1     7     0   104   104   104   104     0     0     0     0     0     0     0     0     0     0

>>
```

It takes a total of 14 cycles to calculate the initial result however each new matrix come directly after the other meaning that the max output will be 1 maxtrix result every 4 cycles, which is 40 ns, and means 250 million results per second assuming no stalling and no delay

Comparison to Project 2

Project 2 took only 11 cycles however there were zeros padding between each set of new coefficients and results were staggered compared to Proj3 implementation which gets all results at once. This means Project 2 has only 200 Million results per second as it takes 5 cycles between every result updating. The Drawback of my design is the resource usages.