Report of Lab 2.7

Lalit Meena,2019CS50439

For this Lab 2 we are building microarchitecture for subset of arm instructions.

This lab involves putting together some stage 1 modules, to form a simple processor which can execute the following subset of ARM instructions with limited variants/features. {add, sub, cmp, mov, ldr, str, beq, bne, b} .Then in stage 3 ,we extend single cycle datapath(stage 2) to flexible and efficient clock period multicycle path.

In stage4 ,we have tested our design exhaustively and reported waves/test-cases for most DP instructions. Then in stage 5 ,we are adding one more features to our implementation which supports shift operations on register and immediate operand variations.In stage 6 ,we are supporting more DT instructions features. It include byte and half word transfers (signed and unsigned), auto increment/decrement with option of pre/post indexing.

Now in stage 7 we are including all variations of multiply instructions with mul_acc module

Zip folder name – L2.7_2019CS50439

Submission folder, **L2.7_2019CS50439** contains-

1. **alu.vhd –** alu module design file
2. **data_mem.vhd -** data memory 128x32 module design file
3. **rfile.vhd -** design file for module register file 16x32
4. **others.vhd-**contains additional modules require for stage-2**(Instruction Decoder, Condition Checker)**
5. **flags.vhd-** design file for module flags **(one line change)**
6. **multicycle.vhd-** SIMPLE multi CYCLE PROCESSOR
7. **shifter.**vhd- Shifter which support 4 shift types-LSL,LSR,ASR,ROR
8. **PM.**vhd- combination circuit path between the processor and memory
9. **Mul_acc.vhd-** module to perform multiply -accumulator computation
10. **package.vhd -**package to declare some types.-etc **(states added)run.do**

testbench files-

testbench_multicycle.vhd - testbench file for module simple multicycle processor

testbench_mulac.vhd - testbench file for mul_acc module-**test_mulacc.jpg**

mul_acc TESTCASES EPWAVE PICS-mtestcase-1,2,3,4,5,6….

mul_acc TESTCASES files- mtestcasen.s, n is 1,2,…..

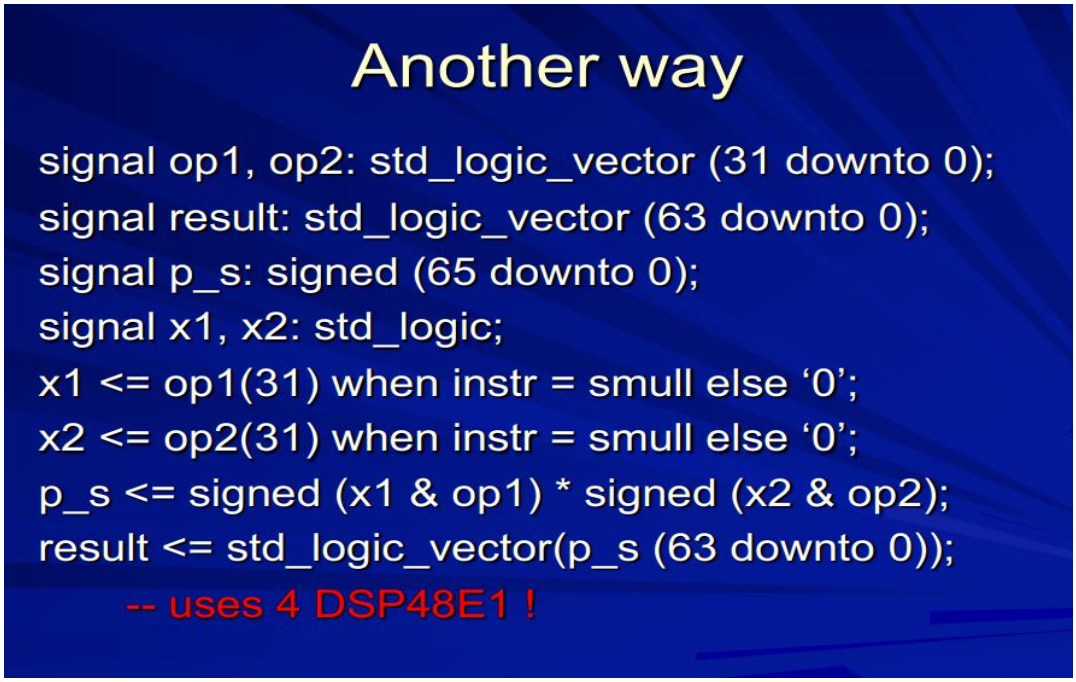
Report of Lab 2.7-lab 2 stage 7 description and test cases(also screenshots)

More details are in next pages of each module-

Mul_acc.vhd – multiply-accumulator module

explanation-

It is combination circuit that performs multiply -accumulator computation.My implementation is based on this another approach discussed in class.-



This is purely combinational circuit, related as r= a*b (+c)

```
entity mulacc is
port(
  a: in word ;
  b: in word;
  c: in std_logic_vector (63 downto 0);--word
  mopc:in std_logic_vector (2 downto 0);
  r: out std_logic_vector (63 downto 0));
```

6 types of multiply and accumulate operation possible ,-

a) mul- “000”multiply with 32-bit output 32x32 => 32

mla- “001”multiply-accumulate with 32-bit output 32x32+32 => 32

b) umull- “100”unsigned multiply with 64-bit output 32x32 => 64

umlal- “101”unsigned multiply-accumuate with 64-bit output 32x32+64 => 64

smull- “110”signed multiply with 64-bit output 32x32 => 64

smlal- “111”signed multiply-accumuate with 64-bit output 32x32+64 => 64

simulated with help of testbench_mulac.vhd-

Here we divided testing into two parts for short and long instructions so,each part can also be run seperately.For comparison among these 6 types of variations(varying mopc signal) we have kept a,b,c input to same complex value,covering possible variation.

Result = a*b (+c)

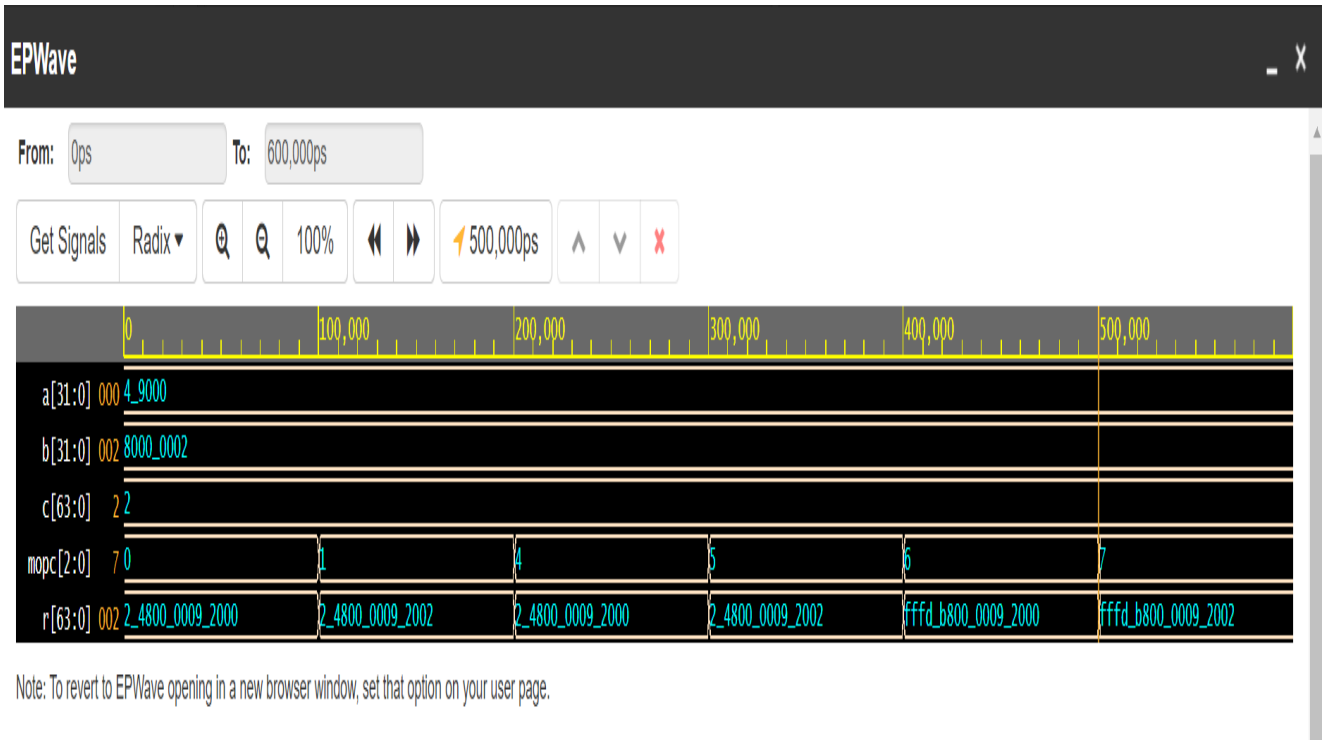
a-#0X00049000

b-#0X80000002

c-#2

We have introduced mopc signal to track currently which of 6 types of variations is running. It is 3 bit signal corresponding to mopc_type: (MUL,MLA,UMULL,UMLAL,SMULL,SMLAL);

test_MULACC-here all six types of variations are tested.



Merging stage 7 in main multicycle processor

MULACC module merge require appropriate adjustment,signals to match its port values to main programme signals.

Number of states used inside main clocked process is states-(fetch,read_AB,arithm,MSTR,MLDR,w2RF,READC,RSHIFT,WB2RF,MULAC,MW2RF).Also used state signal in testbench to show at which state running currently.

At this stage our design have only 11 states ,here we HAVE added two new states –

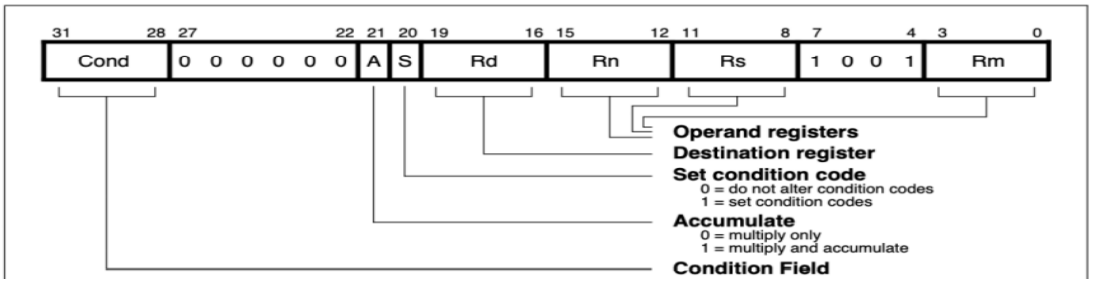
MULAC-to perform multiply and accumulation computation and then settings flags.storing result in 64 bit MLA register.

MW2RF-for simplicity to write agin Register file ,this is used to write RDHIGH register when instruction is of long -64 bit type.here need to indrodue new state to access RF again .

For flags ,we take advantage of wise implementation to result input-MLA(MSB 32) or MLA(lsb)

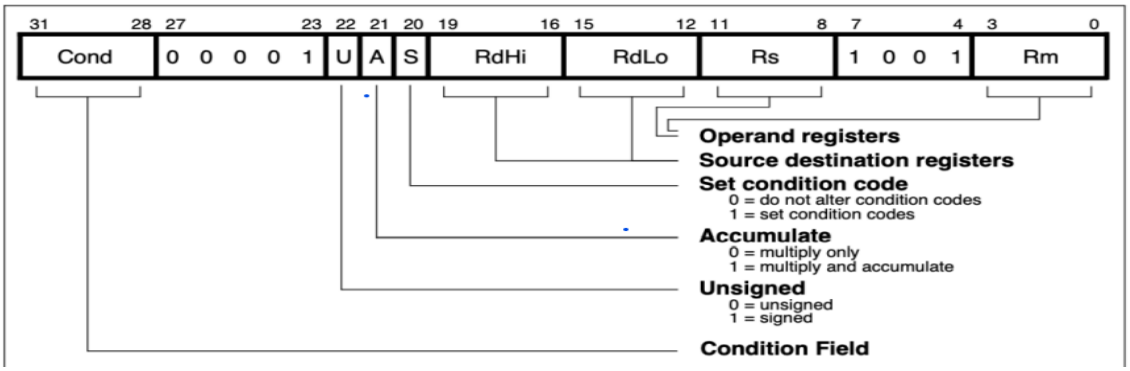
Format of mul_acc Instructions in ARM according to IR(23)-long instruction or not-

`mul:` Rd ← Rm * Rs `mla:` Rd ← Rm * Rs +Rn

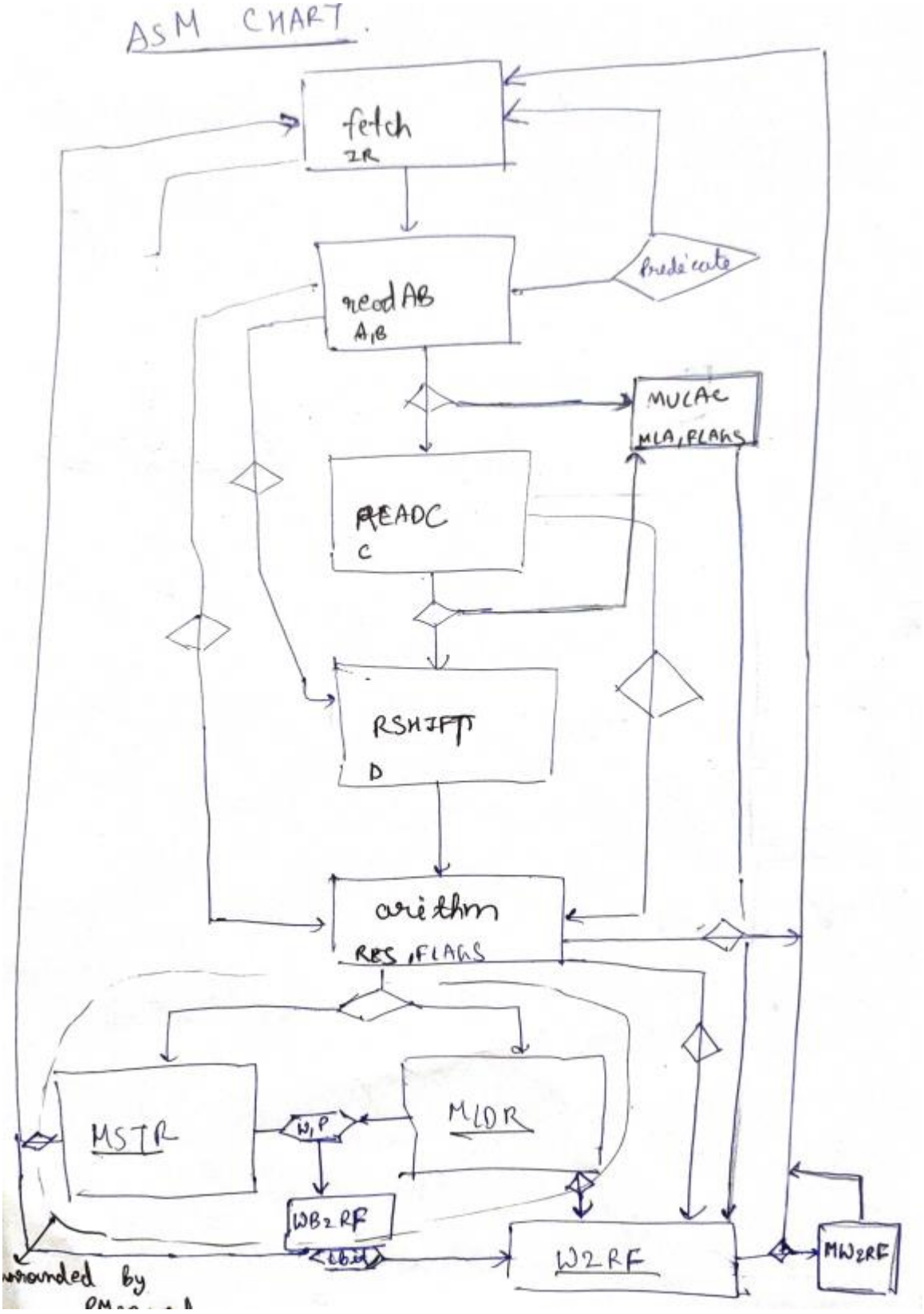


Here long='1'-

`smull,umull:` Rd ← Rm * Rs `smlal,umlal:` Rd ← Rm * Rs + Rd



ASM CHART.


fetch
IR
predicate
readAB
A,B
MULAC
MLA,FLAGS
READC
C
RSHIFT
D
arithm
RES,FLAGS
MSTR
WiP
MLDR
WB2RF
W2RF
MW2RF

Testcases

simulated with help of testbench_multicycle.vhd-

state change like flags and registers at end of program or step can be checked by signal like A,B,C,D,FLAGS, IR,PC,RES,state,minstype ,MWetc. Here main things to look - mopc signal and MLA 64 bit result which then further checked by moiv instruction(B signal).

Correctness can be justified by values of registers read after instructions,etcAlso added extra mov instructions to read and show value of B,RES OR 2nd operand.

We can observe number of cycles etc,also used state signal to show which state currently And also which type of mopc variation.

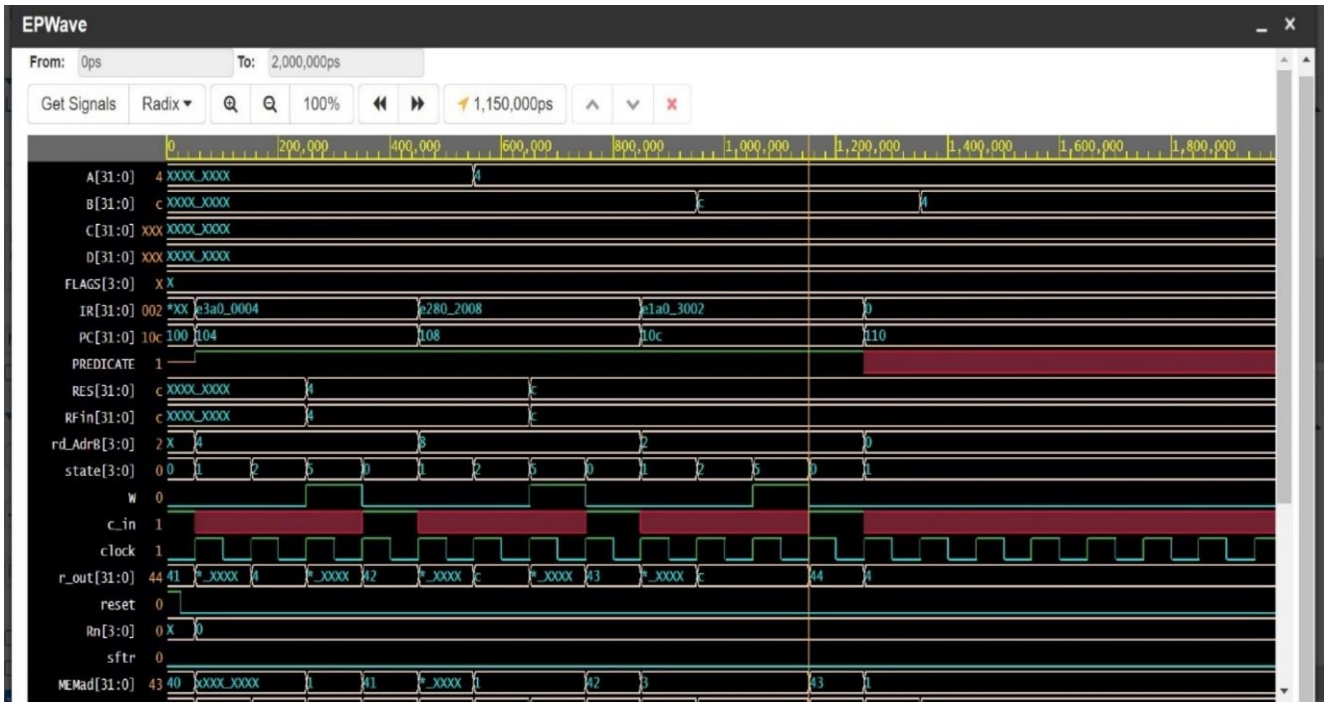
Below are MUL exhaustive related testcases-which covers all variants multiply and accumulate instruction possible in MUL.

We can identify MUL_ACC related parameters in middle-end -mopc(type),MLA(64 bit result) ,etc.

Take help of **run time limit** on simulator for taking required PC increments.For clock related testbench signals generation and this limit can sometime generate error like pipe broken(resolve by changing).On checking again runtime limit canbe set to epwave top listed value.

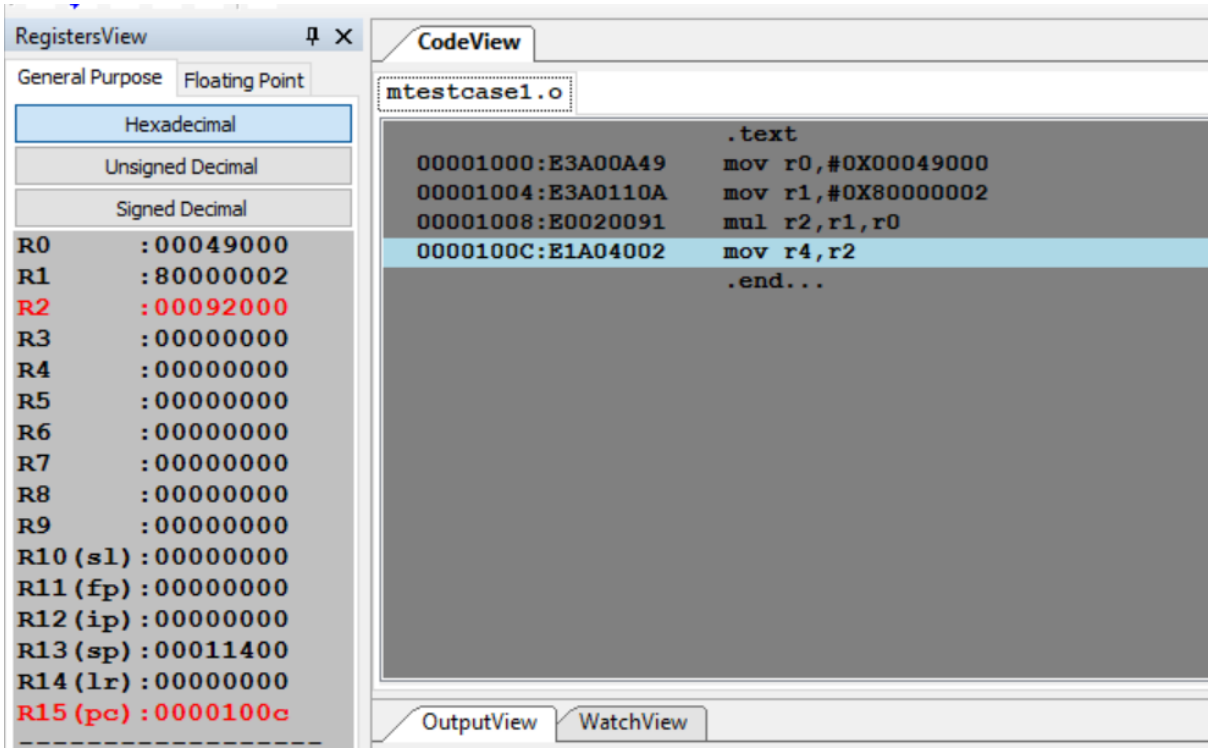
TESTCASES SEPARATE EPWAVE PICS and assembly files CAN also BE FOUND IN SUBMITTED FOLDER.

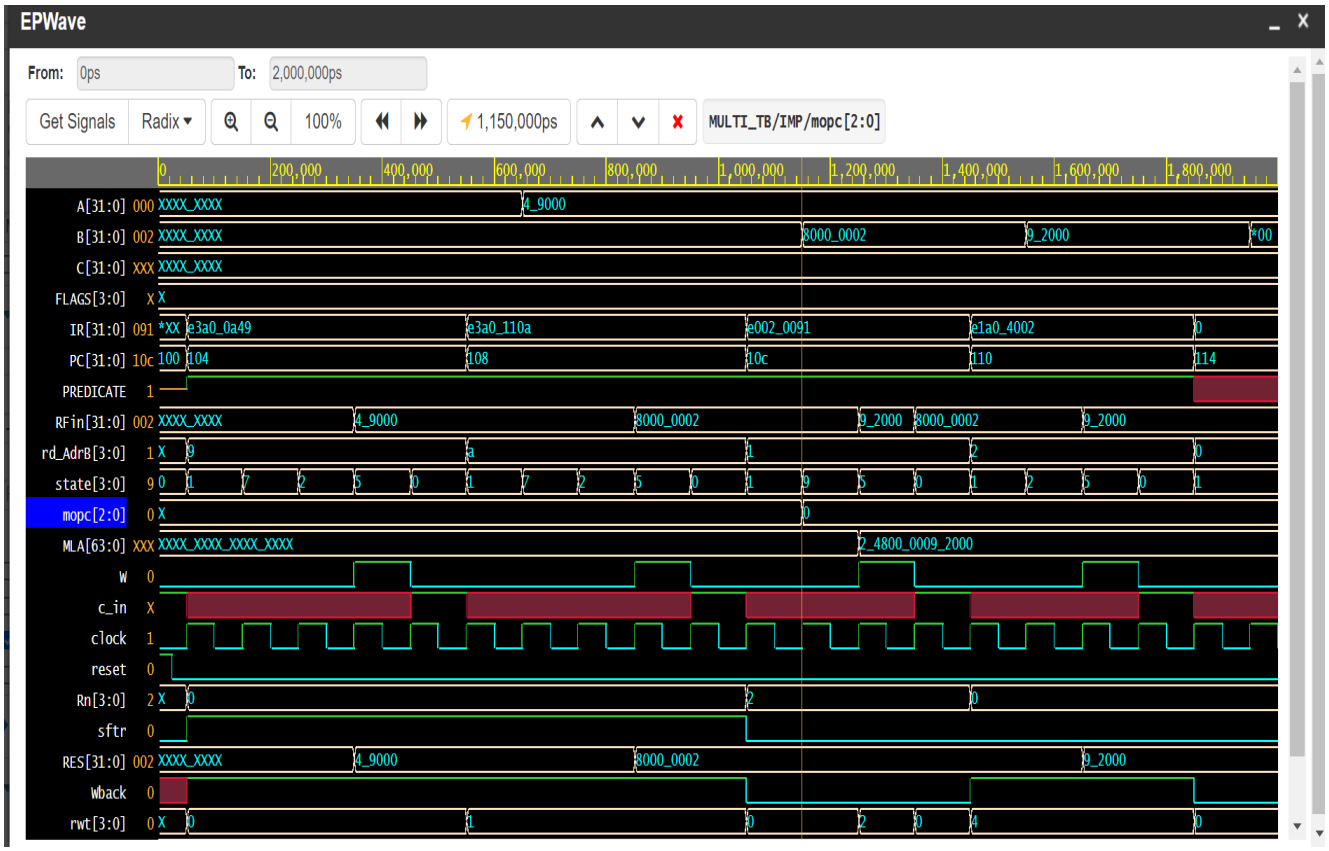
Check with previous testcase-2,found that previous implementation is working same for previous testcases.



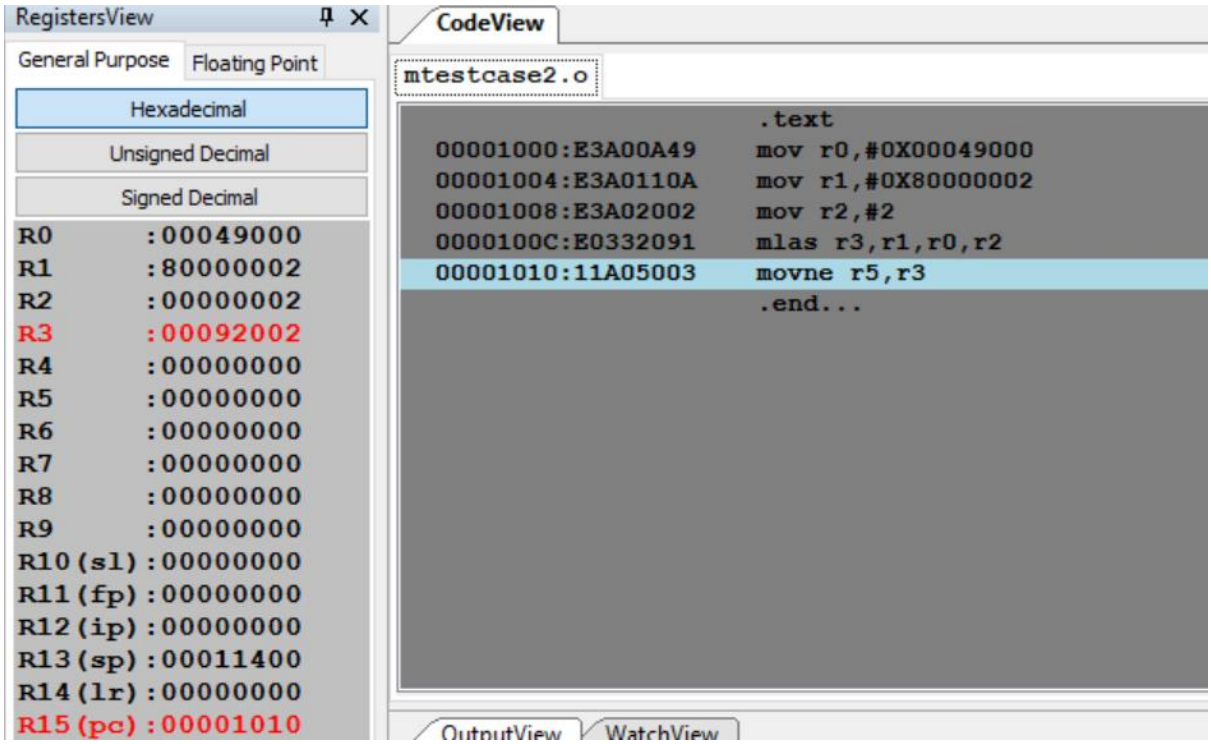
Also working same for previous pmconnect stage pmtestcase1

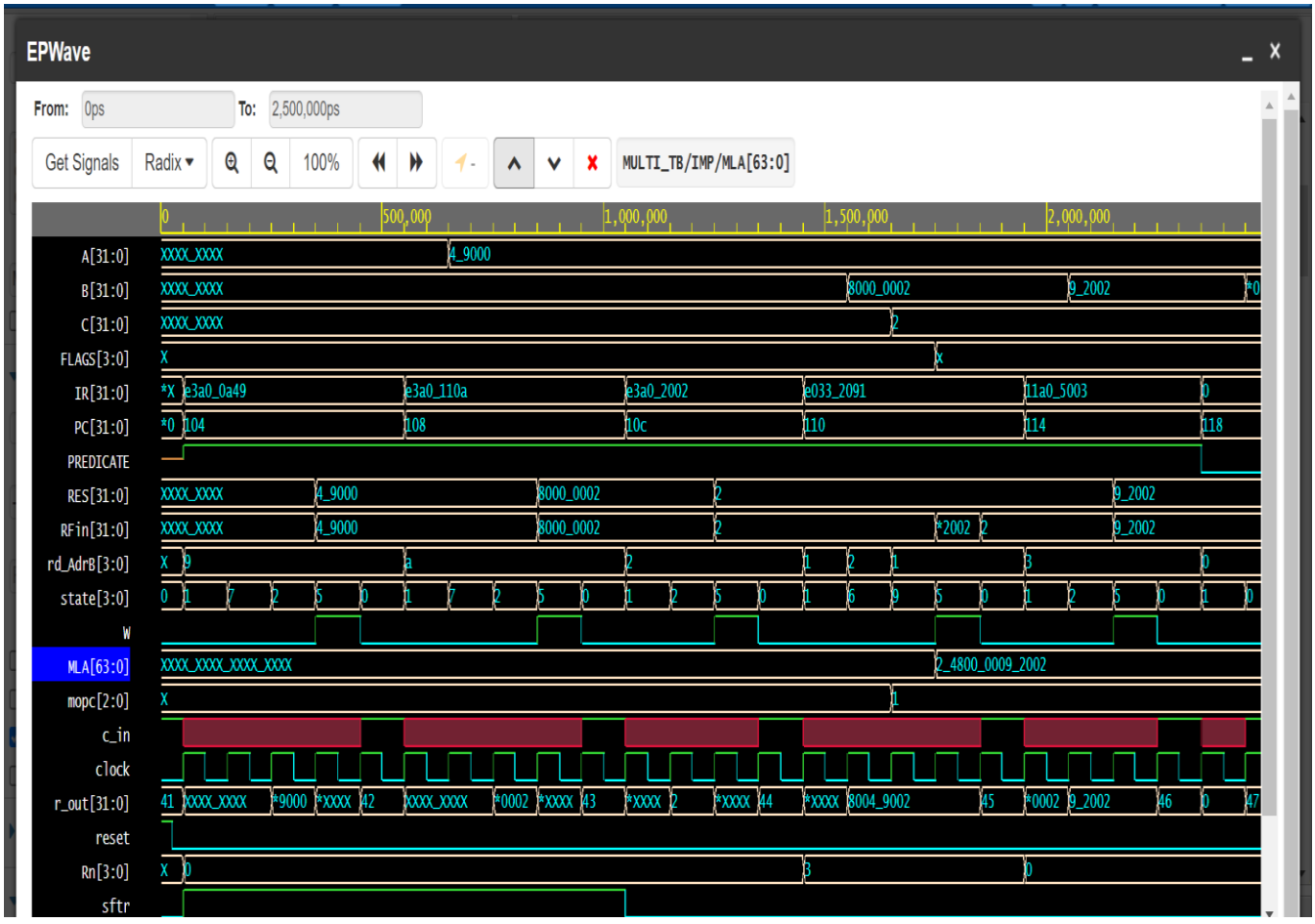
Mtestcase1


RegistersView
General Purpose
Floating Point
Hexadecimal
Unsigned Decimal
Signed Decimal
R0 :00049000
R1 :80000002
R2 :00092000
R3 :00000000
R4 :00000000
R5 :00000000
R6 :00000000
R7 :00000000
R8 :00000000
R9 :00000000
R10(sl):00000000
R11(fp):00000000
R12(ip):00000000
R13(sp):00011400
R14(lr):00000000
R15(pc):0000100c
CodeView
mtestcase1.o
.text
00001000:E3A00A49 mov r0,#0X00049000
00001004:E3A0110A mov r1,#0X80000002
00001008:E0020091 mul r2,r1,r0
0000100C:E1A04002 mov r4,r2
.end...
OutputView
WatchView

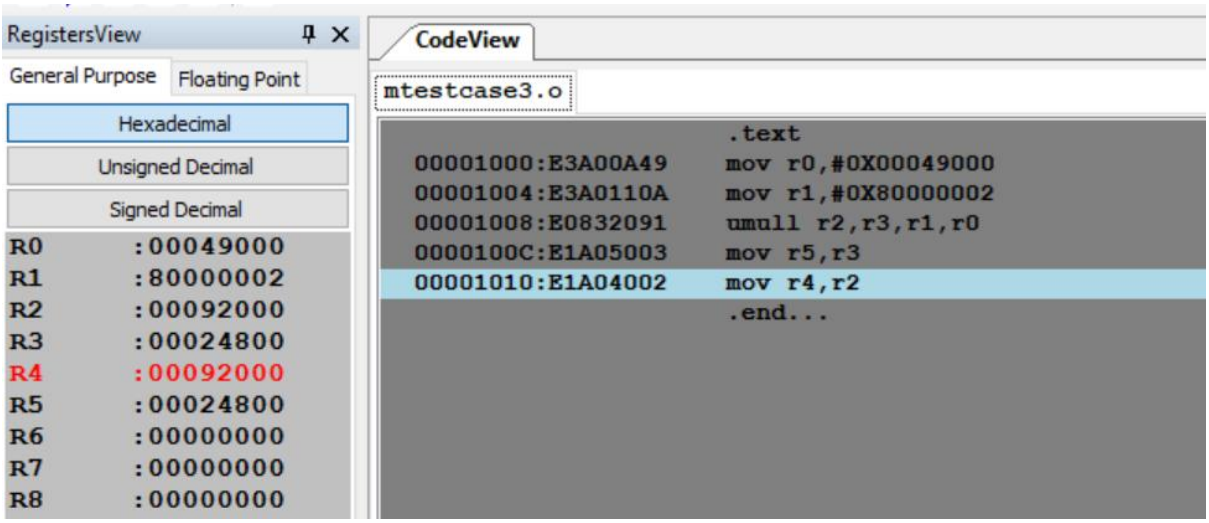

EPWave
From: 0ps
To: 2,000,000ps
Get Signals
Radix
100%
1,150,000ps
MULTI_TB/IMP/mopc[2:0]
A[31:0]
B[31:0]
C[31:0]
FLAGS[3:0]
IR[31:0]
PC[31:0]
PREDICATE
RFin[31:0]
rd_AdrB[3:0]
state[3:0]
mopc[2:0]
MLA[63:0]
W
c_in
clock
reset
Rn[3:0]
sftr
RES[31:0]
Wback
rwt[3:0]

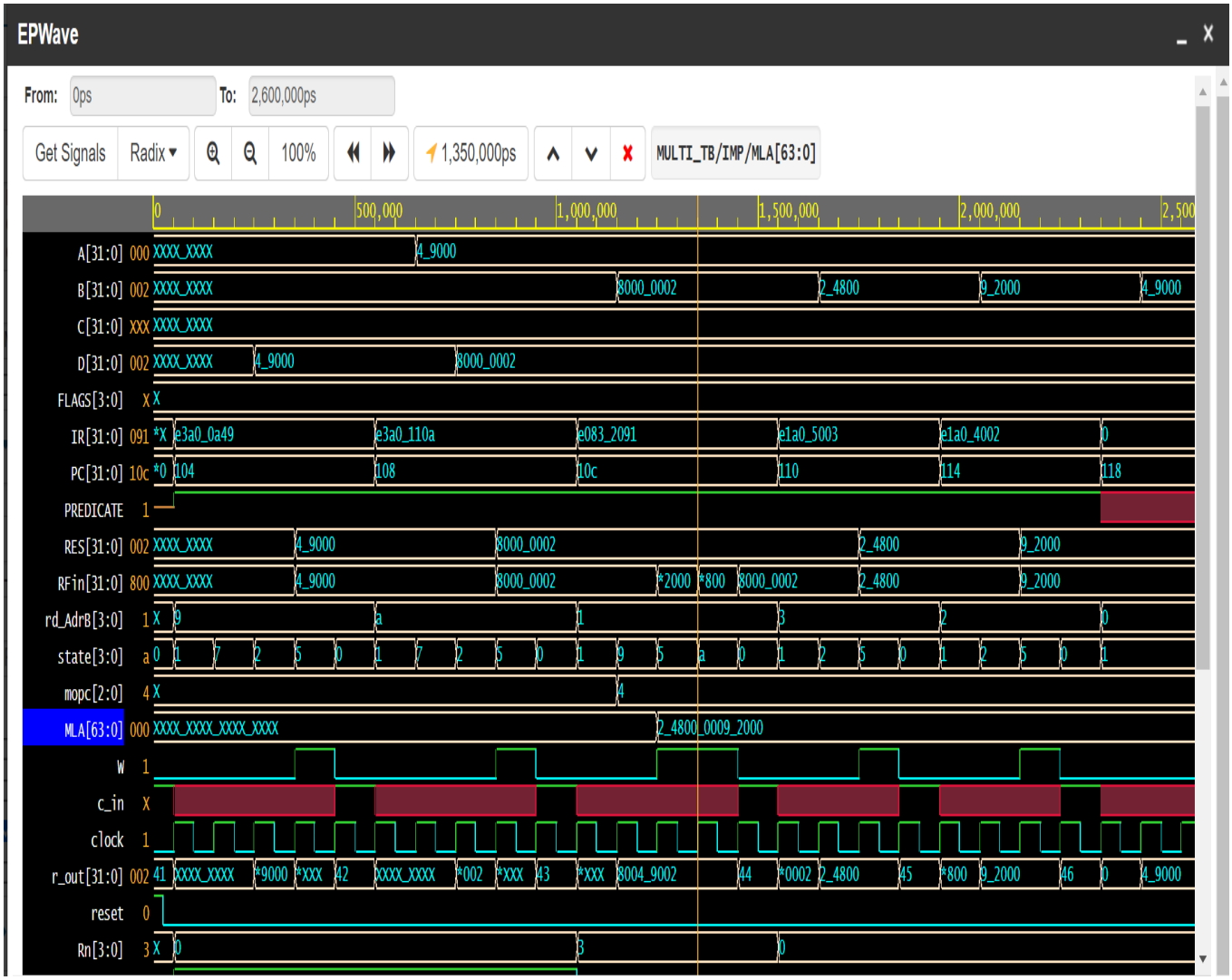
Mtestcase2


RegistersView
General Purpose
Floating Point
Hexadecimal
Unsigned Decimal
Signed Decimal
R0 :00049000
R1 :80000002
R2 :00000002
R3 :00092002
R4 :00000000
R5 :00000000
R6 :00000000
R7 :00000000
R8 :00000000
R9 :00000000
R10(sl) :00000000
R11(fp) :00000000
R12(ip) :00000000
R13(sp) :00011400
R14(lr) :00000000
R15(pc) :00001010
CodeView
mtestcase2.o
.text
00001000:E3A00A49 mov r0,#0X00049000
00001004:E3A0110A mov r1,#0X80000002
00001008:E3A02002 mov r2,#2
0000100C:E0332091 mlas r3,r1,r0,r2
00001010:11A05003 movne r5,r3
.end...
OutputView
WatchView

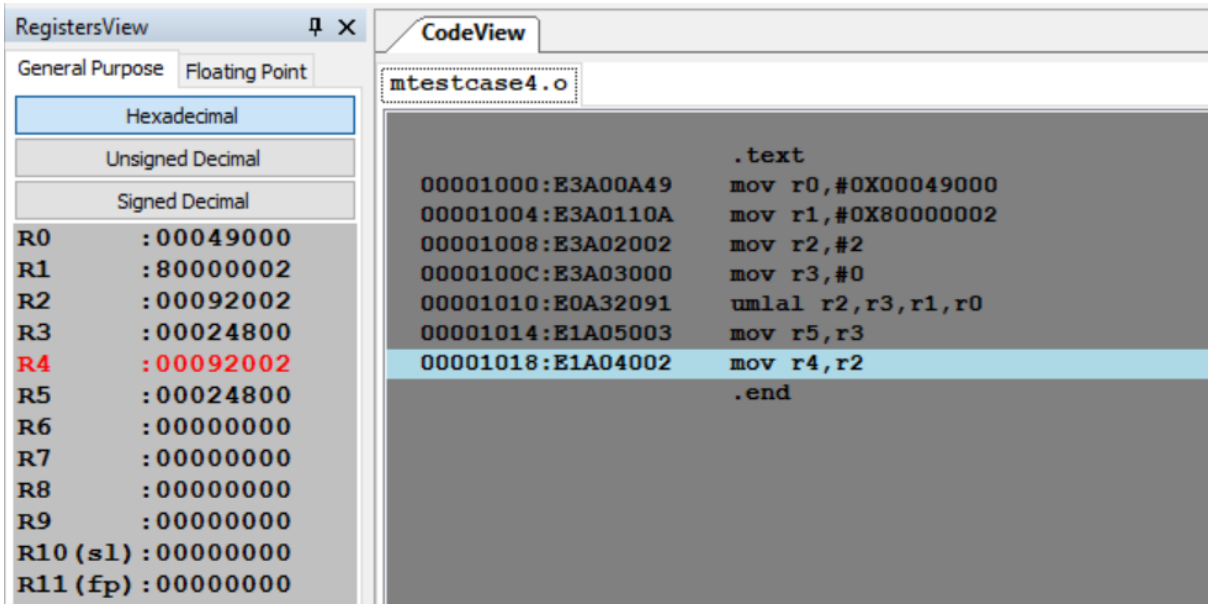

EPWave
From: 0ps
To: 2,500,000ps
Get Signals
Radix
100%
MULTI_TB/IMP/MLA[63:0]
A[31:0]
B[31:0]
C[31:0]
FLAGS[3:0]
IR[31:0]
PC[31:0]
PREDICATE
RES[31:0]
RFin[31:0]
rd_Adr8[3:0]
state[3:0]
W
MLA[63:0]
mopc[2:0]
c_in
clock
r_out[31:0]
reset
Rn[3:0]
sftr

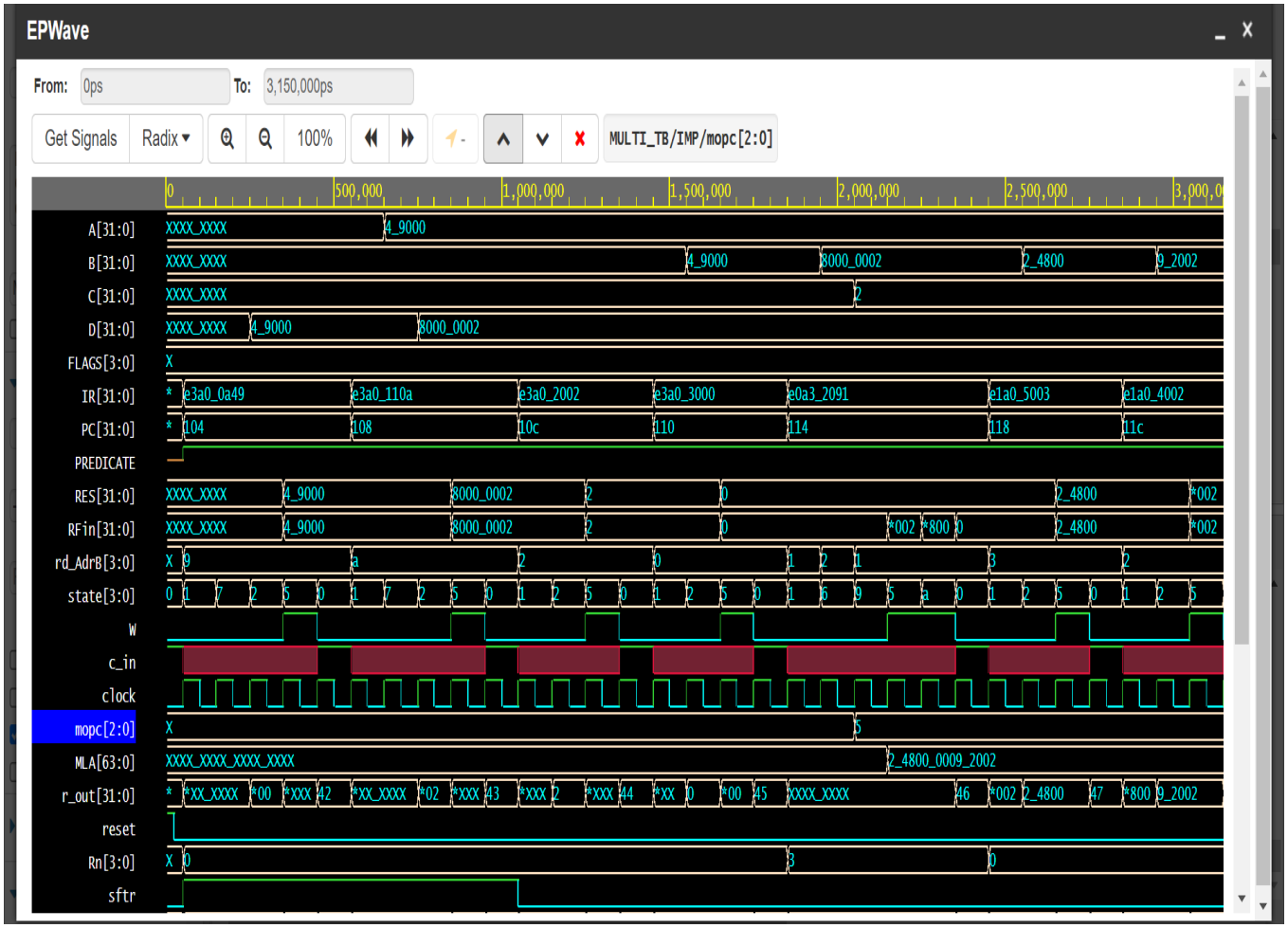
Mtestcase3


RegistersView
General Purpose
Floating Point
Hexadecimal
Unsigned Decimal
Signed Decimal
R0 :00049000
R1 :80000002
R2 :00092000
R3 :00024800
R4 :00092000
R5 :00024800
R6 :00000000
R7 :00000000
R8 :00000000
CodeView
mtestcase3.o
.text
00001000:E3A00A49 mov r0,#0X00049000
00001004:E3A0110A mov r1,#0X80000002
00001008:E0832091 umull r2,r3,r1,r0
0000100C:E1A05003 mov r5,r3
00001010:E1A04002 mov r4,r2
.end...

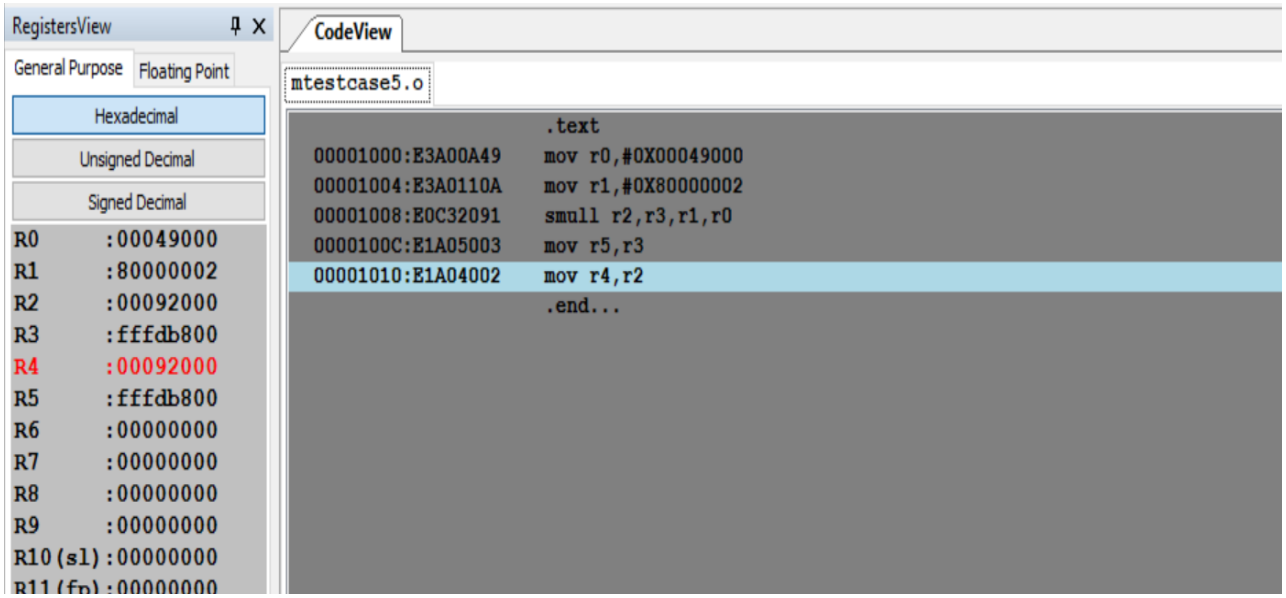

EPWave
From: 0ps
To: 2,600,000ps
Get Signals
Radix
100%
1,350,000ps
MULTI_TB/IMP/MLA[63:0]
A[31:0]
B[31:0]
C[31:0]
D[31:0]
FLAGS[3:0]
IR[31:0]
PC[31:0]
PREDICATE
RES[31:0]
RFin[31:0]
rd_AdrB[3:0]
state[3:0]
mopc[2:0]
MLA[63:0]
W
c_in
clock
r_out[31:0]
reset
Rn[3:0]

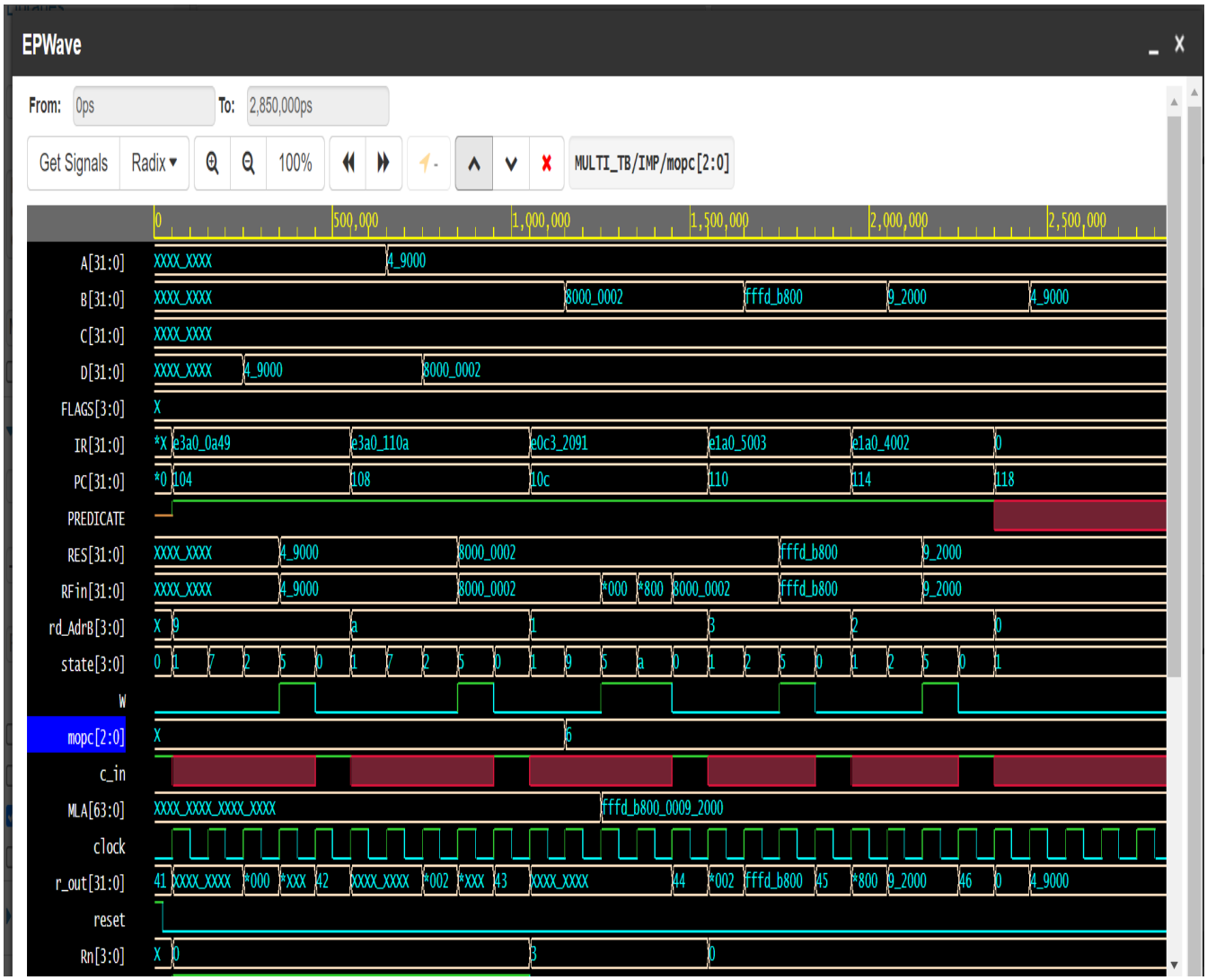
Mtestcase4


RegistersView
General Purpose
Floating Point
Hexadecimal
Unsigned Decimal
Signed Decimal
R0 :00049000
R1 :80000002
R2 :00092002
R3 :00024800
R4 :00092002
R5 :00024800
R6 :00000000
R7 :00000000
R8 :00000000
R9 :00000000
R10(sl):00000000
R11(fp):00000000
CodeView
mtestcase4.o
.text
00001000:E3A00A49 mov r0,#0X00049000
00001004:E3A0110A mov r1,#0X80000002
00001008:E3A02002 mov r2,#2
0000100C:E3A03000 mov r3,#0
00001010:E0A32091 umlal r2,r3,r1,r0
00001014:E1A05003 mov r5,r3
00001018:E1A04002 mov r4,r2
.end

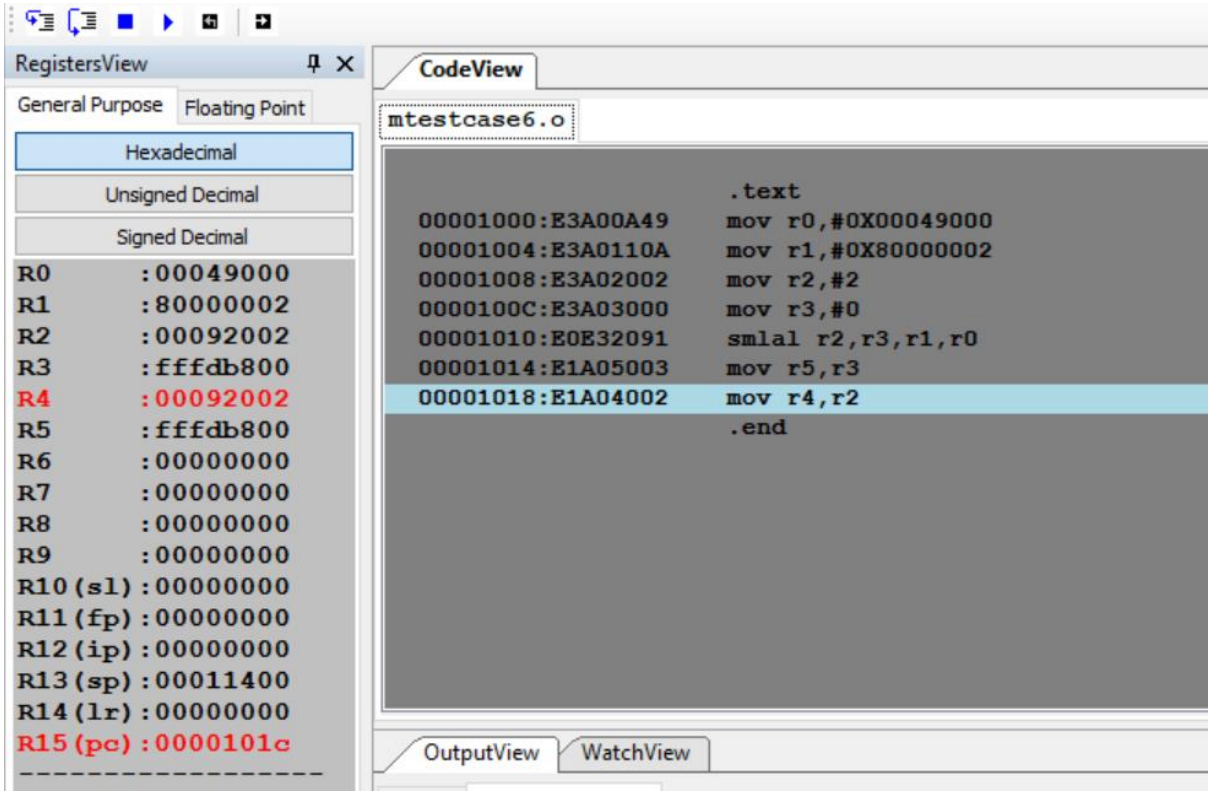

EPWave
From: 0ps
To: 3,150,000ps
Get Signals
Radix
100%
MULTI_TB/IMP/mopc[2:0]
A[31:0]
B[31:0]
C[31:0]
D[31:0]
FLAGS[3:0]
IR[31:0]
PC[31:0]
PREDICATE
RES[31:0]
RFin[31:0]
rd_AdrB[3:0]
state[3:0]
W
c_in
clock
mopc[2:0]
MLA[63:0]
r_out[31:0]
reset
Rn[3:0]
sftr

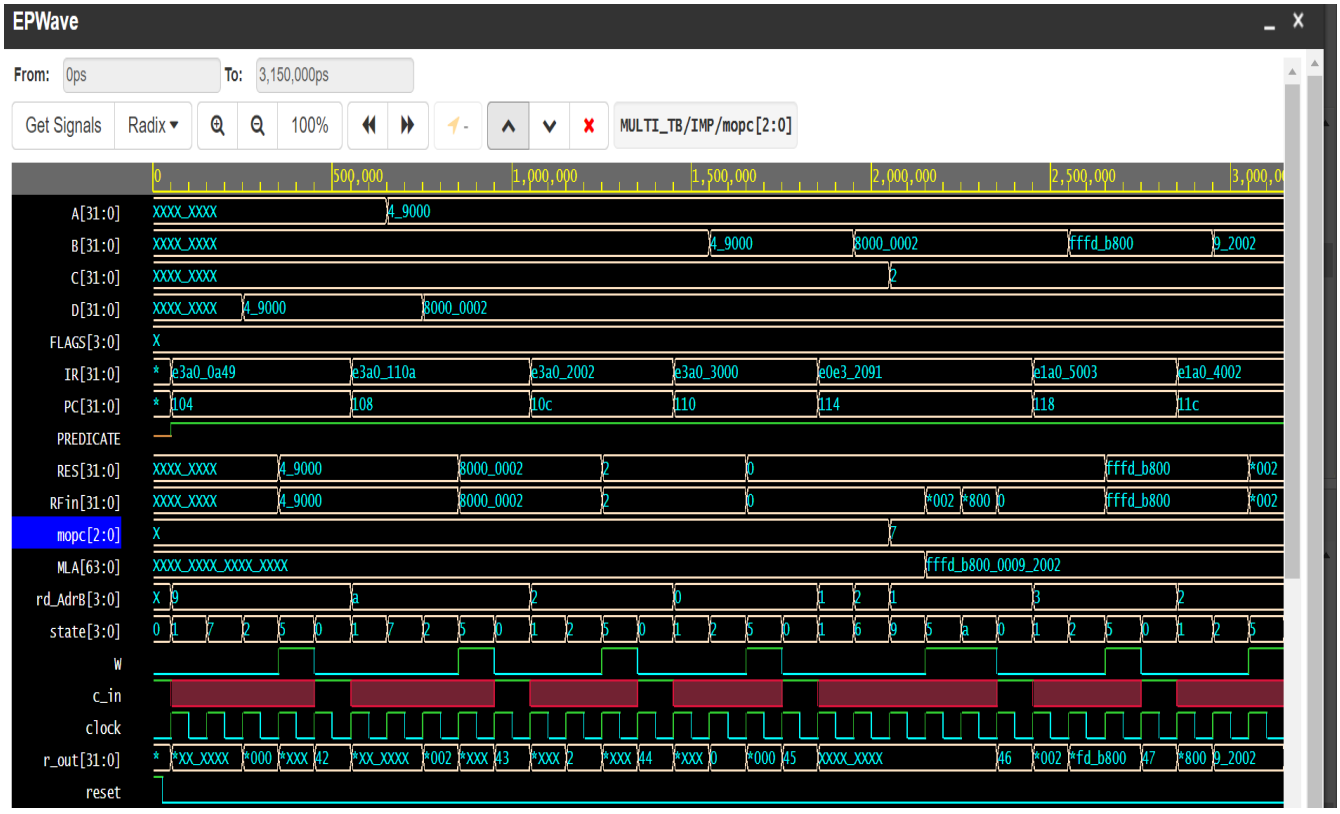
Mtestcase5


RegistersView
General Purpose
Floating Point
Hexadecimal
Unsigned Decimal
Signed Decimal
R0 :00049000
R1 :80000002
R2 :00092000
R3 :fffdb800
R4 :00092000
R5 :fffdb800
R6 :00000000
R7 :00000000
R8 :00000000
R9 :00000000
R10 (sl) :00000000
R11 (fp) :00000000
CodeView
mtestcase5.o
.text
00001000:E3A00A49 mov r0,#0X00049000
00001004:E3A0110A mov r1,#0X80000002
00001008:E0C32091 smull r2,r3,r1,r0
0000100C:E1A05003 mov r5,r3
00001010:E1A04002 mov r4,r2
.end...


EPWave
From: 0ps
To: 2,850,000ps
Get Signals
Radix
100%
MULTI_TB/IMP/mopc[2:0]
A[31:0]
B[31:0]
C[31:0]
D[31:0]
FLAGS[3:0]
IR[31:0]
PC[31:0]
PREDICATE
RES[31:0]
RFin[31:0]
rd_AdrB[3:0]
state[3:0]
W
mopc[2:0]
c_in
MLA[63:0]
clock
r_out[31:0]
reset
Rn[3:0]

Mtestcase6


RegistersView
General Purpose
Floating Point
Hexadecimal
Unsigned Decimal
Signed Decimal
R0 :00049000
R1 :80000002
R2 :00092002
R3 :fffdb800
R4 :00092002
R5 :fffdb800
R6 :00000000
R7 :00000000
R8 :00000000
R9 :00000000
R10(sl):00000000
R11(fp):00000000
R12(ip):00000000
R13(sp):00011400
R14(lr):00000000
R15(pc):0000101c
CodeView
mtestcase6.o
.text
00001000:E3A00A49 mov r0,#0X00049000
00001004:E3A0110A mov r1,#0X80000002
00001008:E3A02002 mov r2,#2
0000100C:E3A03000 mov r3,#0
00001010:E0E32091 smlal r2,r3,r1,r0
00001014:E1A05003 mov r5,r3
00001018:E1A04002 mov r4,r2
.end
OutputView
WatchView


EPWave
From: 0ps
To: 3,150,000ps
Get Signals
Radix
100%
MULTI_TB/IMP/mopc[2:0]
A[31:0]
B[31:0]
C[31:0]
D[31:0]
FLAGS[3:0]
IR[31:0]
PC[31:0]
PREDICATE
RES[31:0]
RFin[31:0]
mopc[2:0]
MLA[63:0]
rd_AdrB[3:0]
state[3:0]
W
c_in
clock
r_out[31:0]
reset

For each module,work and analysis I have put files in edaplayground like below-

Used for simulation- Aldec riviera pro 2020.04

simulation settings For eg mtestcase1-

For more testcases and live running ,edaplayground can be played in demo for this stage and public link of it can be shared.