Report of Lab 2.3

Lalit Meena,2019CS50439

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

This stage involves putting together 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} .Here we extend single cycle datapath to flexible and efficient clock period multicycle path.

In previous modules ,made almost no changes in them but only extended size of DATA Memory module changed .

Here in comparison previous stage 2,there is only changes happened in main processor design file and testbench file.

Zip folder name – L2.2_2019CS50439

Submission folder, **L2.1_2019CS50439** contains-

8. **alu.vhd –** alu module design file
9. **data_mem.vhd -** data memory 128x32 module design file
10. **rfile.vhd -** design file for module register file 16x32
11. **others.vhd-**contains additional modules require for stage-2**(Instruction Decoder, Condition Checker)**
12. **flags.vhd-** design file for module flags
13. **multicycle.vhd-** SIMPLE multi CYCLE PROCESSOR

package.vhd -package to declare some types-words,bytes,etc

run.do- edaplayground file to help in syntheses of modules to give useful info about resources

testbench files-

testbench_multicycle.vhd - testbench file for module simple multicycle processor

TESTCASES PICS-testcase-1,2….

Report of Lab 2.3-lab 2 stage 3 description and test cases(also screenshots)

More details are in next pages of each module-

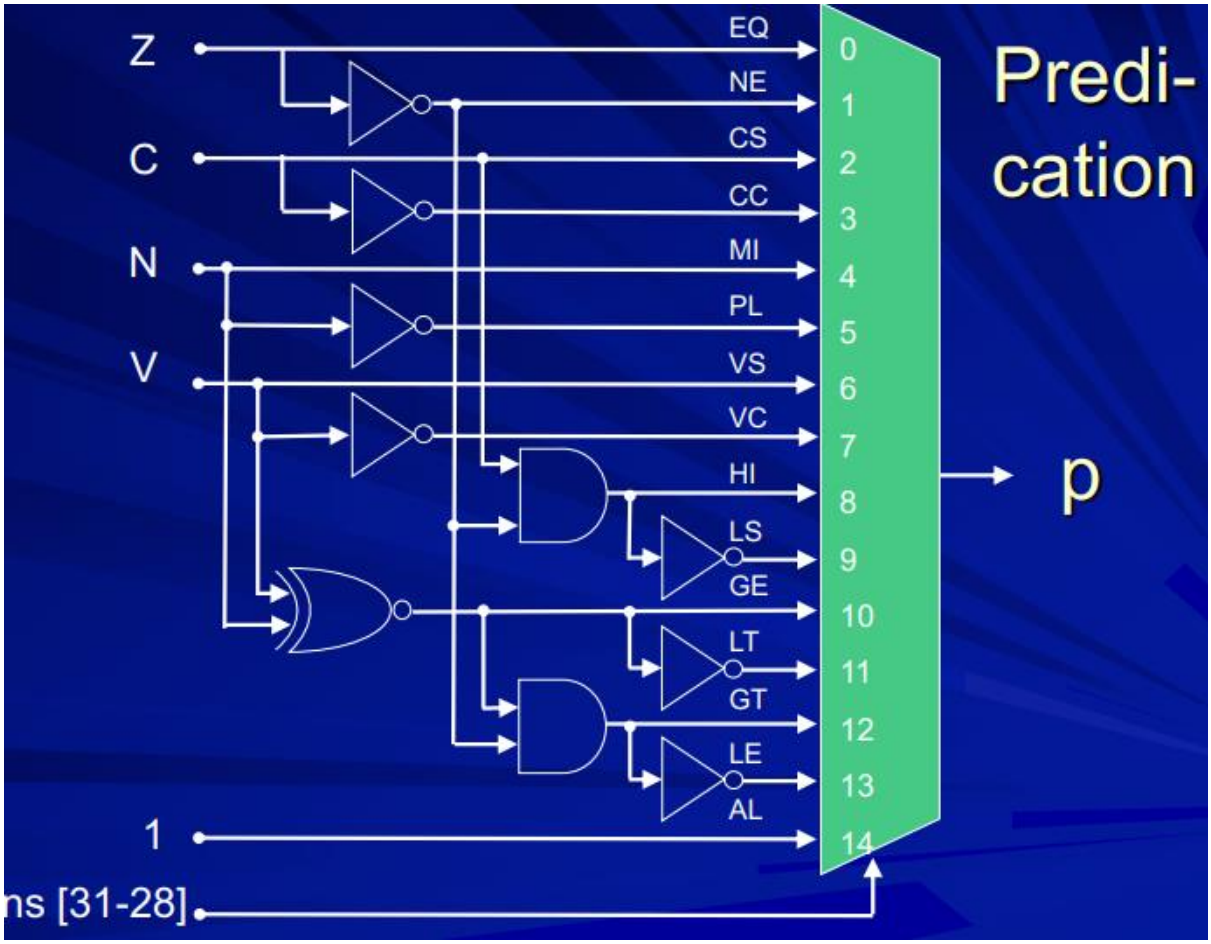
11. others.vhd – multiple module design file

explanation-

This file contains contains additional modules require for stage-2**(Instruction Decoder, Condition Checker)**

Decoder-its like combinational control path which takes fetched instruction from PMem and generates control signals for main processor data path and other components

Condition Checker- This is a combinational circuit that looks at the Flags and condition field of the instruction to decide whether the specified condition is true or not-predicate result. Here now designed for EQ and NE conditions which are relevant for this stage, this is a simple circuit and works on below logic-



Components and ports info-

```
COMPONENT Decoder is
    Port (
        instruction : in word;
        instr_class : out instr_class_type;
        operation : out optype;
        DP_subclass : out DP_subclass_type;
        DP_operand_src : out DP_operand_src_type;
        load_store : out load_store_type;
        cond: out nibble;
        DT_offset_sign : out DT_offset_sign_type
    );
end COMPONENT Decoder;


 COMPONENT pcr IS
 PORT(
     NXT : IN word;
     -- write when R is 1, else "ZZZZZZZ
     reset,PW,CLK : IN STD_LOGIC;
     CURRENT : OUT word
     );
 END COMPONENT pcr;

 COMPONENT CHECKER IS
 PORT(
     FLAGS : IN nibble;
     COND : IN nibble;
     PREDICATE : OUT STD_LOGIC
     );
 END COMPONENT CHECKER;
```

12.**flags.vhd** –flags

explanation-

It have an edge triggered 4-bit flags register (or 4 edge triggered flipflops) that will be set according to required conditions and clocked.

Flags-cvzn can be read unclocked.

Logic and implementation is same as given material uploaded on “Circuit for maintaining Flags”

How DP instructions affect Flags C, V, Z, N

Instructions	Effect on Flags		
	if S-bit = 0	if S-bit = 1 and no shift/rotate	if S-bit = 1 and shift/rotate is there
add, sub, rsb, adc, sbc, rsc	No flags are affected	All 4 flags are affected, ALU carry is used	
cmp, cmn	All 4 flags are affected, ALU carry is used		
and, orr, xor, bic, mov, mvn	No flags are affected	Only Z and N are affected	C, Z and N are affected shift/rotate carry is used
tst, teq	Only Z and N are affected		

entity and ports info-

```
-- entity
entity flag is
port(
    instr_class : in instr_class_type;
    opc:in optype;
    DP_subclass : IN DP_subclass_type;
    s,shiftr:in std_logic;
    c_alu,c_shiftr,CLK:in std_logic;
    a,b:in std_logic;--msb bits
    result: in word;
    cvzn:out nibble);
end entity;
```

13.**Simple.vhd-** SIMPLE SINGLE CYCLE PROCESSOR

Required Assumptions has been taken as stated and taught.

explanation-

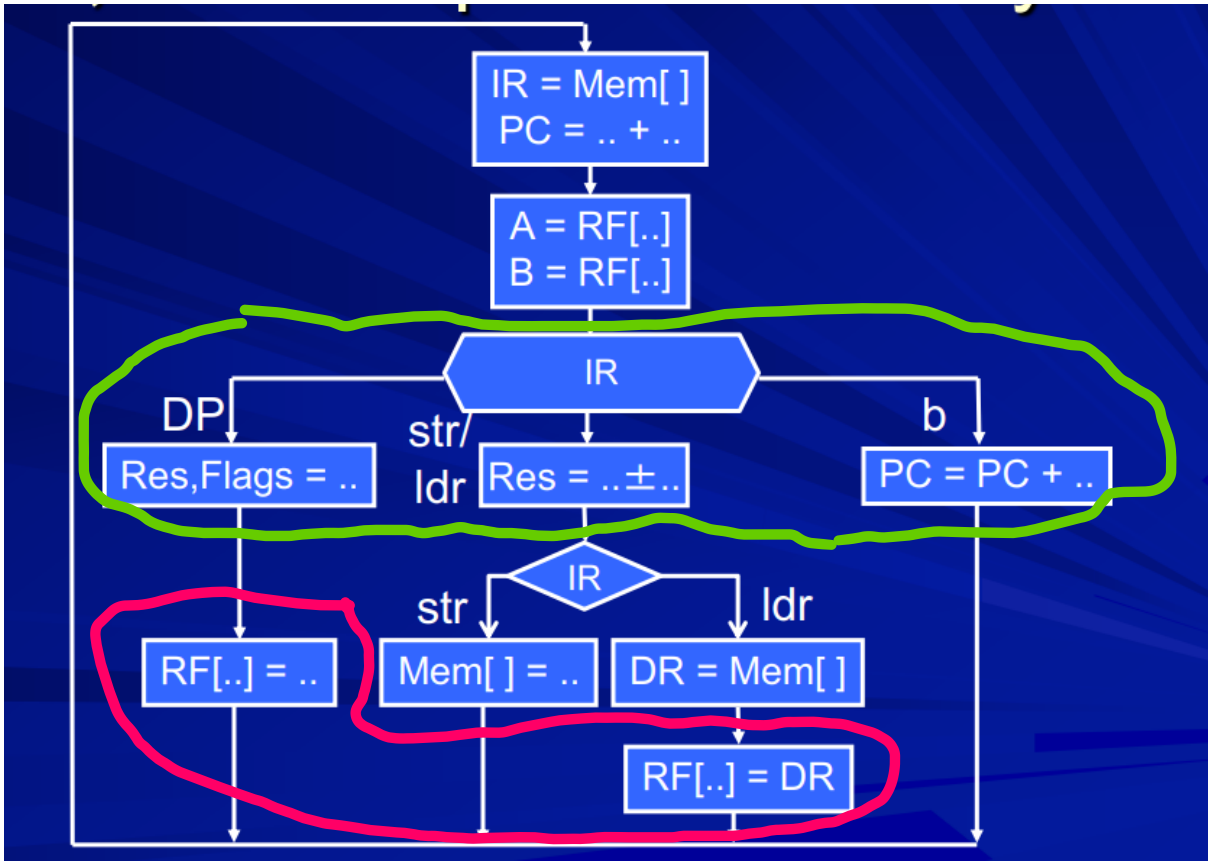
It is main file which have data path for simple single cycle processor.With help of Decoder -control circuit we able to complete required processor. This also include all the "glue logic" which in this case includes some multiplexers and control logic.

Here all required and stated components are initiated,which are all the state elements (register file, data memory, flags, program counter and many more)

This file contains concurrent combinational statements like component instantiations,selecting required parameters,etc

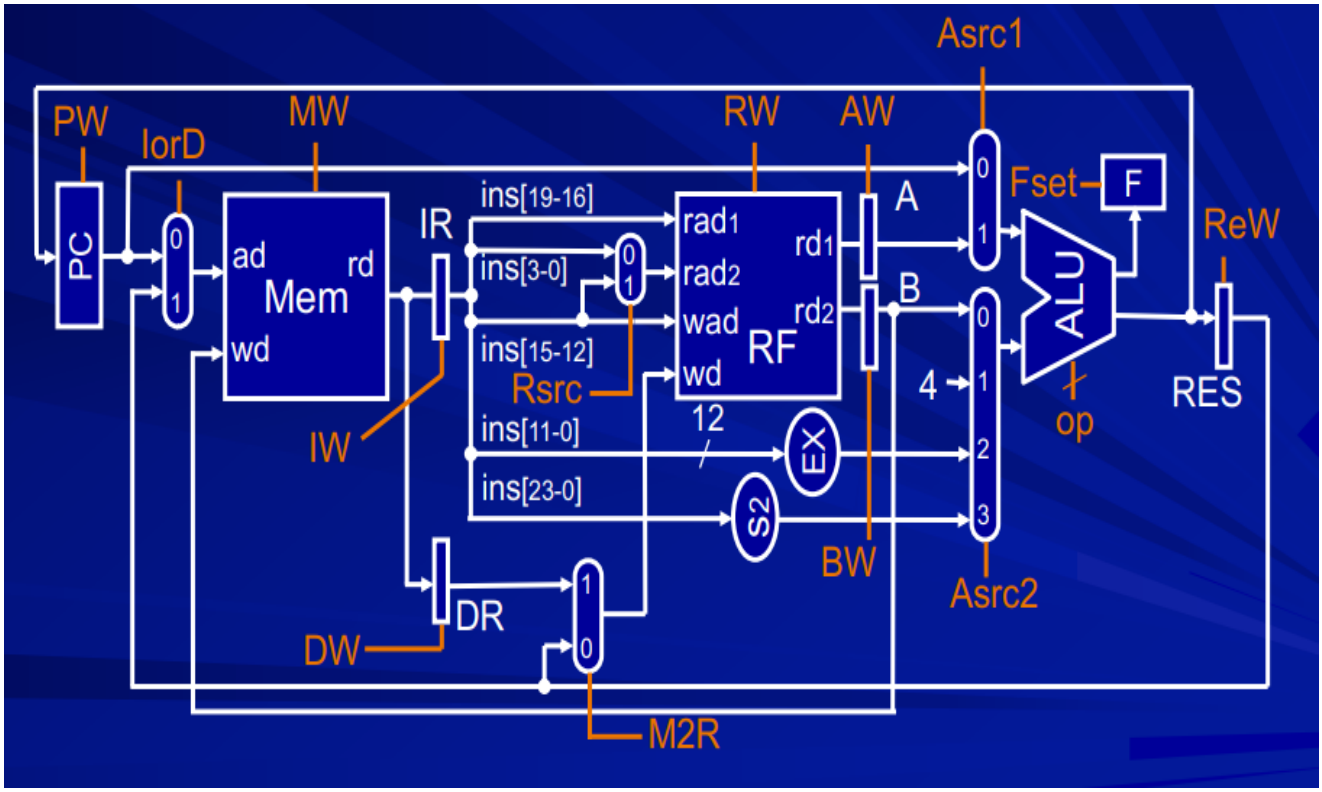
Here evey cycle one fetched instruction by PC get executed.PC get increment conditionally by driven clock.

ASM CHART-



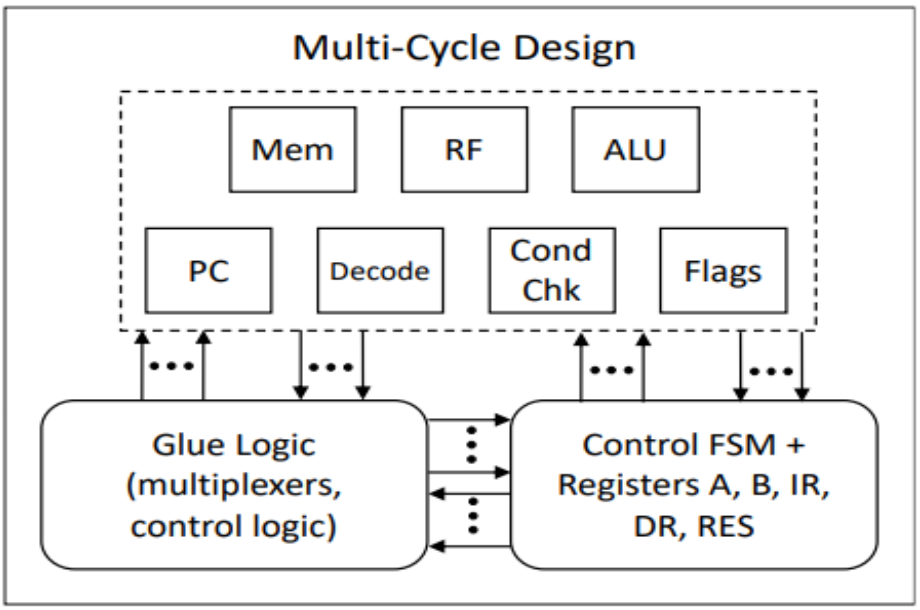
Number of states used inside main clocked process is states- (fetch,read_AB,arithm(green one),MSTR,MLDR,w2RF(pink one)).Also used state signal in testbench to show at which state running currently.

MULTICYCLE DATAPATH WITH CONTROL SIGNALS



For each state in combinational process we generate glue logic and multiplex signal,write enables etc.

USED Registers A, B, IR, DR and RES as local signals,and modify value directly according to states so not need of write enable on them.



entity and ports info-

```
entity processor is
 port (
 clock, reset : in std_logic
 );
end processor;
```

PC

For branch instructions Used word addresses and offsets, rather than byte addresses and offsets, to ALU, by PC(31 downto 2) + word offset + 1. In 1st cycle done by ALU, specifying the operation as adc (add with carry) and making carry input as '1'. Even PC + 4 can be done in the same way in 3rd cycle , that is PC(31 downto 2) + 0 + 1. Word offset is bits 23 downto 0 of the instructions and will need sign extensionby 8 bits. PC(31 downto 2) needs to be extended by two zeros. The word addresses produced by ALU then later converted into byte addresses (logical left shift by 2 positions) before writing into PC.

Used PC as local signal to main body.

DATA MEMORY

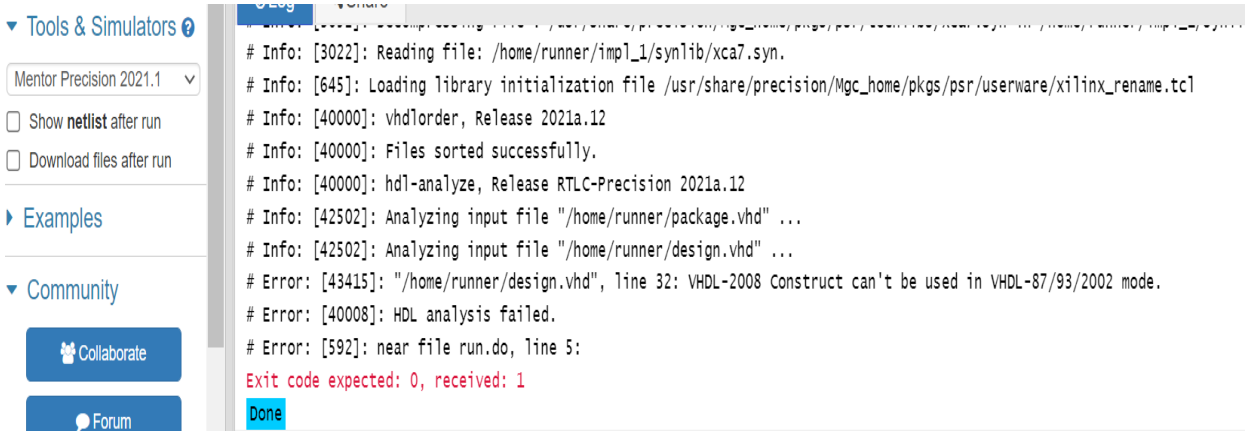
Used one array logically partitioned into two. This would form a memory with one partition "write-protected". Here program memory part are from 64-127 location in array.On reset PC Set to start-64

TO put program ,used method like stage 2 ,initializing in the component as array.But can also be written by initial testbench cycles

Synthesis related

In Stage-2 ,used some new concurrent type of statements like when-else,with-select.This may be reason for encountering this message while synthesis for these using told run.do(Xilinx -family Artix-7 -part 7A100TCSG324) file for Mentor Precision 2021.

Also read on Piazza By sir that synthesis is not necessary.



<u>simulated with help of testbench_multicycle.vhd-</u>

state change like flags and registers at end of program or step can be checked by signal like A,B,FLAGS, IR,PC,RESetc.

Correctness can be justified by values of registers read after instructions,etc

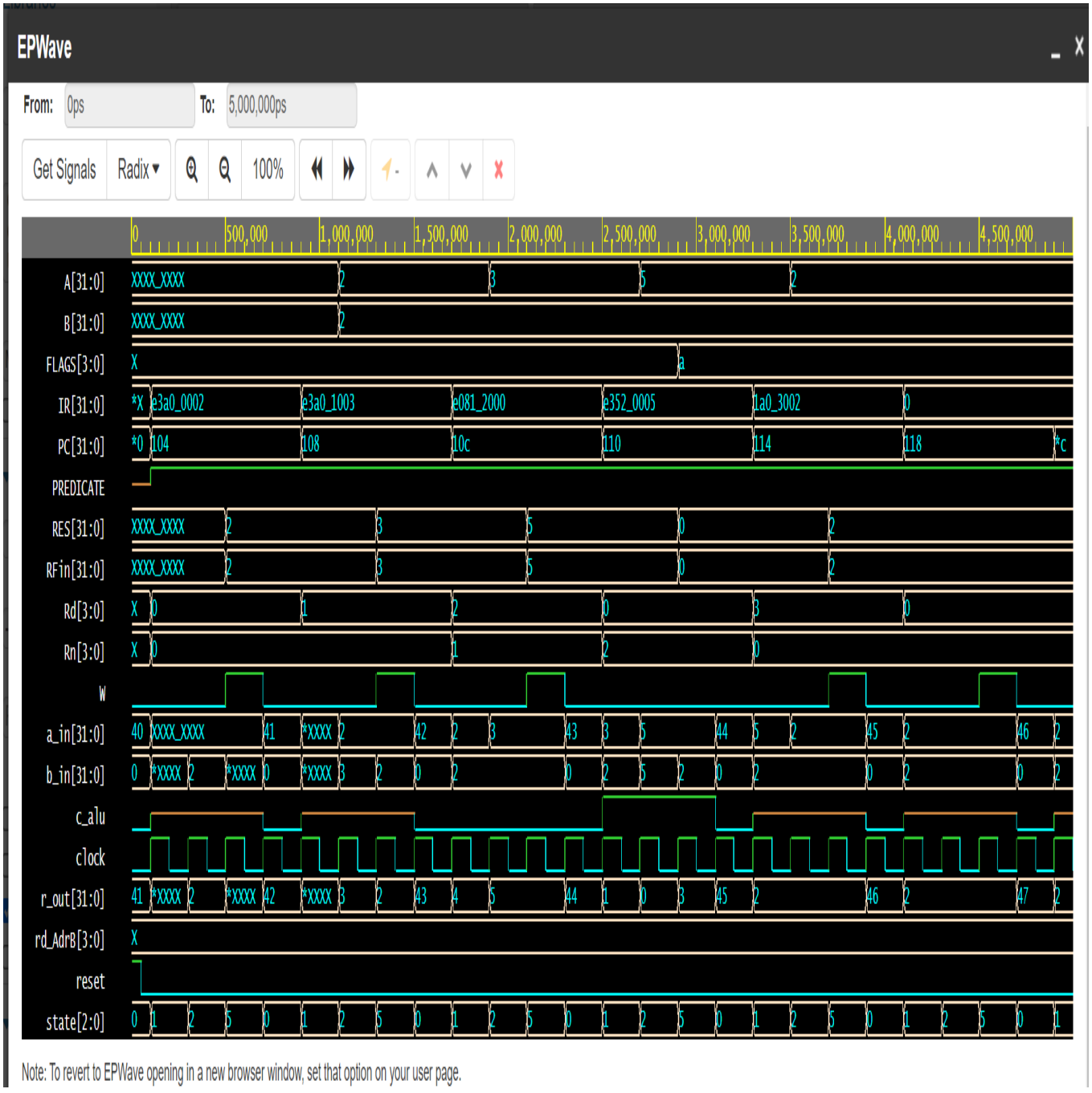
We can observe number of cycles etc,also used state signal in testbench to show at which state running currently.

Take help of **run time limit** on simulator for taking required PC increments.

TESTCASES SEPARATE SIGNAL PICS CAN BE FOUND IN SUBMITTED FOLDER

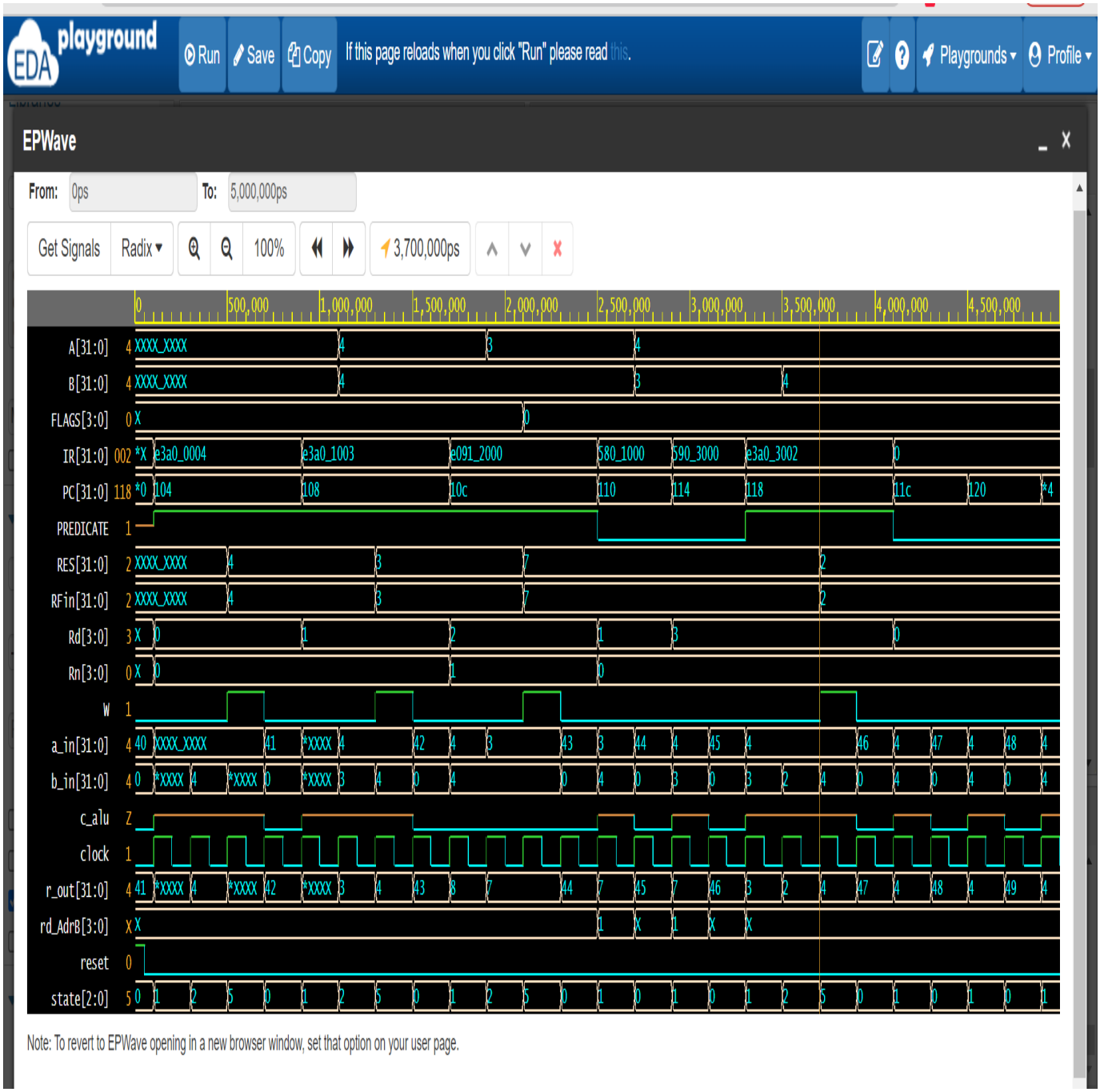
Example-1

```
                    .text
00:E3A00002     mov r0, #2
04:E3A01003     mov r1, #3
08:E0812000     add r2, r1, r0
0C:E3520005     cmp r2, #5
10:01A03002     moveq r3,r2
```



Example-2

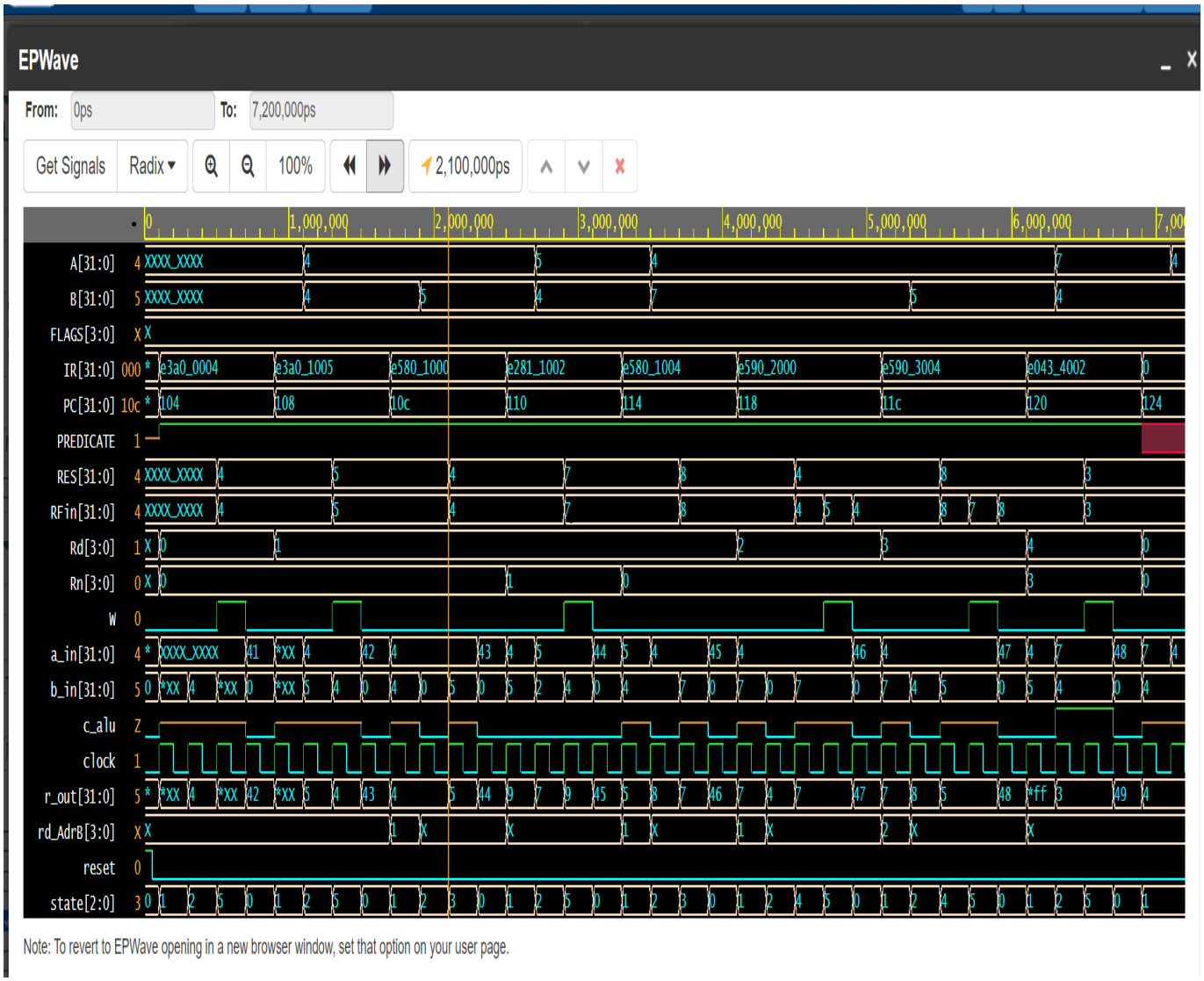
```
                    .text
000:E3A00004    mov r0, #4
004:E3A01003    mov r1, #3
008:E0912000    adds r2, r1, r0
00C:05801000    streq r1,[r0]
010:05903000    ldreq r3,[r0]
014:E3A03002    mov r3,#2
```



Example-3

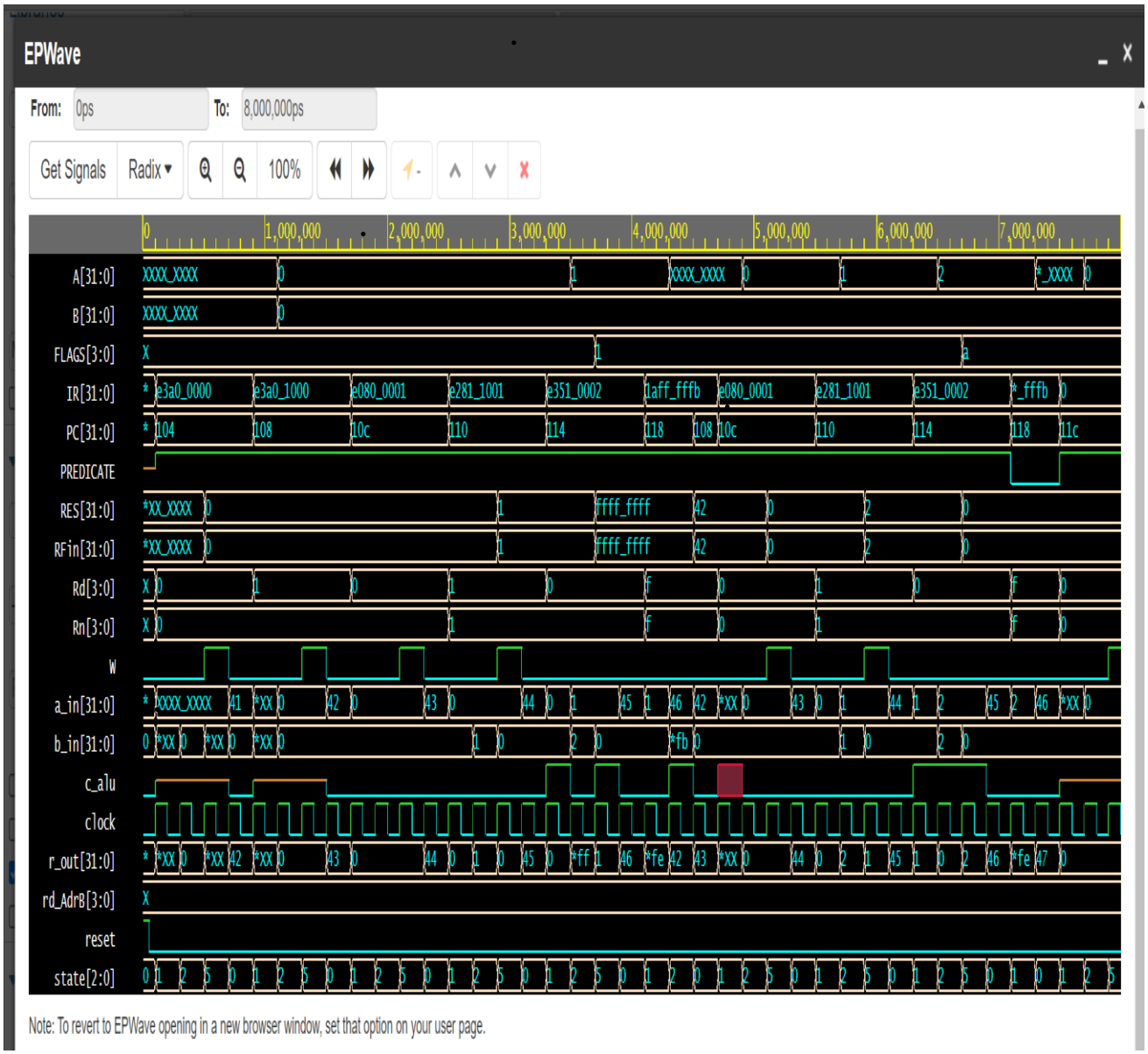
```
signal mem_array : memory_type :=
                   (0 => X"E3A0000A",
                    1 => X"E3A01005",
                    2 => X"E5801000",
                    3 => X"E2811002",
                    4 => X"E5801004",
                    5 => X"E5902000",
                    6 => X"E5903004",
                    7 => X"E0434002",
                    others => X"00000000"
                   );
```

```
.text
mov r0, #10
mov r1, #5
str r1, [r0]
add r1, r1, #2
str r1, [r0, #4]
ldr r2, [r0]
ldr r3, [r0, #4]
sub r4, r3, r2
.end
```



Example-4

signal mem_array : memory_type :=
(0 => X"E3A00000",
1 => X"E3A01000",
2 => X"E0800001",
3 => X"E2811001",
4 => X"E3510003",
5 => X"1AFFFFFB",
others => X"00000000"
);
	.text
	mov r0, #0
	mov r1, #0
	Loop: add r0, r0, r1
	add r1, r1, #1
	cmp r1, #3
	bne Loop
	.end

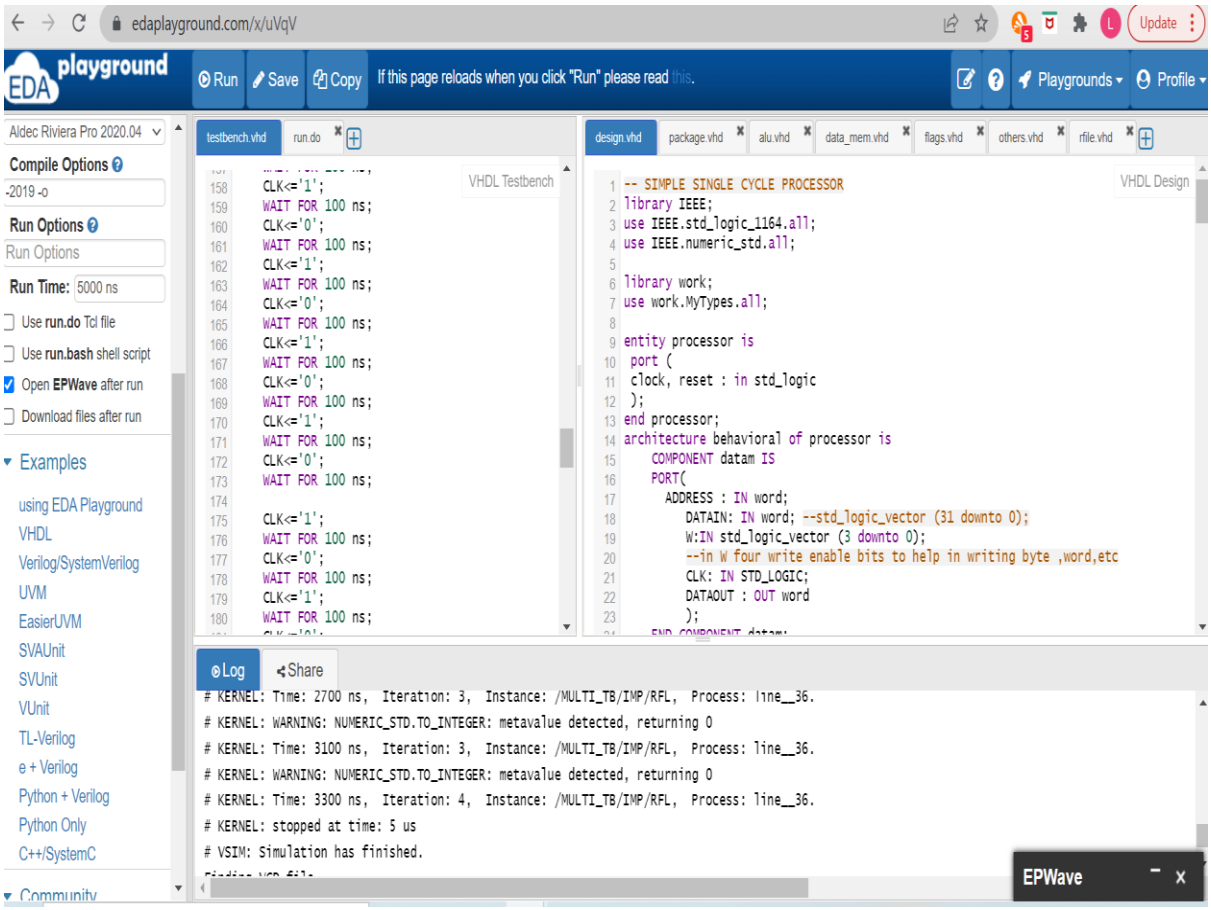


For each module,work and analysis I have put files in edaplayground like below-

Used for simulation- Aldec riviera pro 2020.04

Used for synthesis- Mentor Precision 2021.1

simulation settings For eg testcase-2-



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