

## Report of Lab 2.6

**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.

Now 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.

### **Zip folder name – L2.6\_2019CS50439**

Submission folder, **L2.6 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
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. **package.vhd** -package to declare some types-words,bytes,etc **run.do**

### **testbench files-**

**testbench\_multicycle.vhd** - testbench file for module simple multicycle processor

**testbench\_PMconnect.vhd** - testbench file for P-M path module

**P-M path module testing PICS-** test\_pmconnect\_str & test\_pmconnect\_ldr

**P-M path TESTCASES EPWAVE PICS-pmtestcase-1,2,3,4,5,6....**

**P-M path TESTCASES files-** pmtestcasen.s, n is 1,2,.....

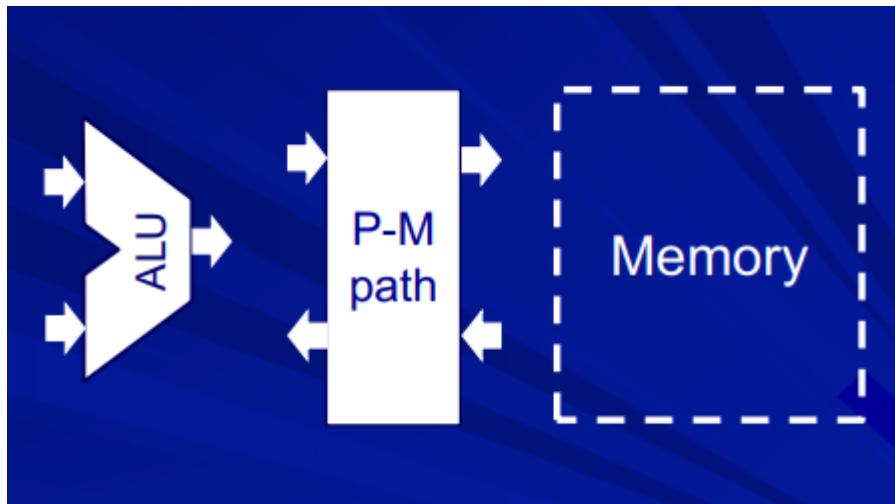
**Report of Lab 2.6-lab 2 stage 6 description and test cases(also screenshots )**

**More details are in next pages of each module-**

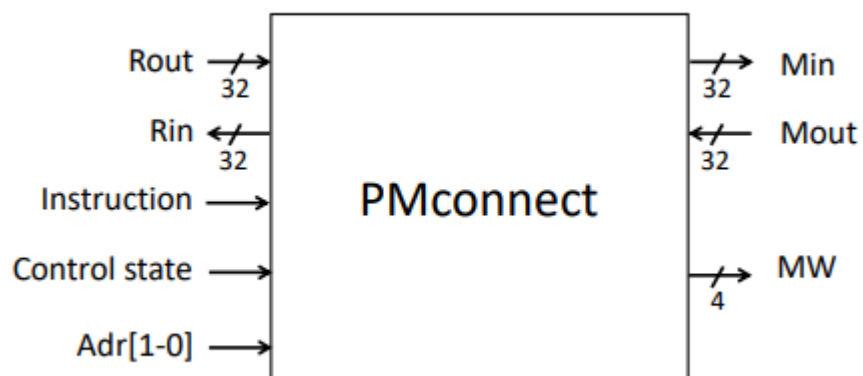
## PM.vhd – P-M path

### explanation-

It is combination circuit between the processor and memory which does the required transformation of words into half-words / bytes and vice versa.



This is purely combinational circuit.



entity and ports info-

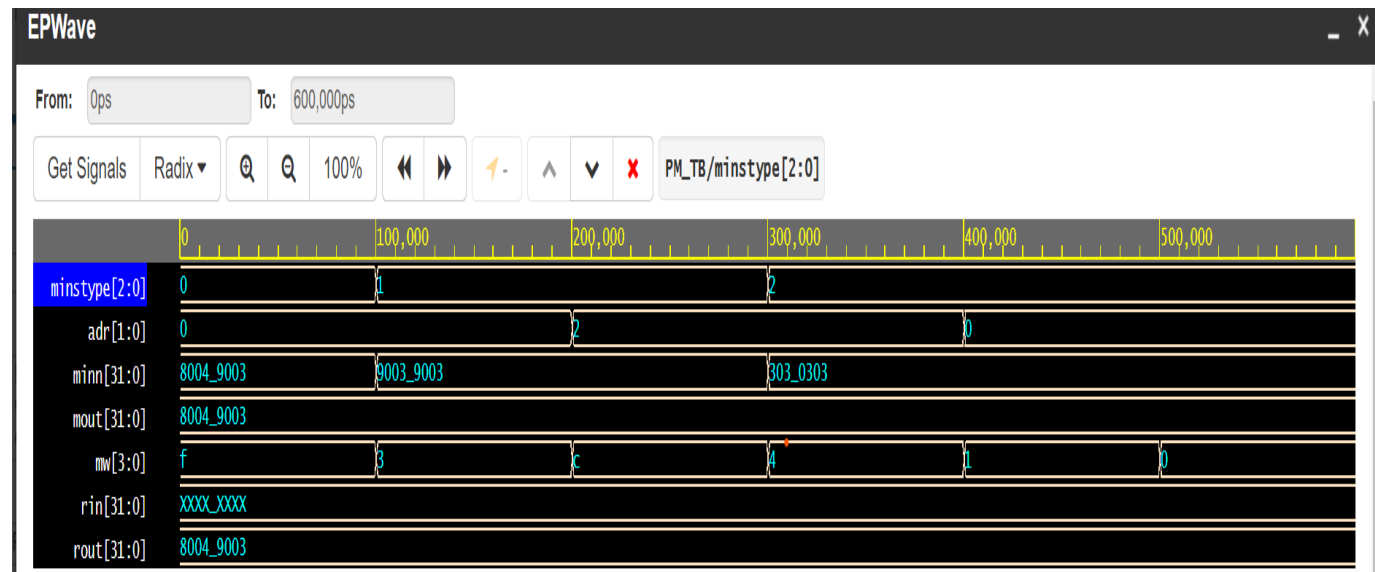
```
entity PMconnect is
port(
  minstr: in minstr_type ;
  rout: in word;
  mout: in word;
  cstate: in ctrl_state; --use ?
  adr: in std_logic_vector (1 downto 0);
  minn: out word;
  rin : out word;
  mw : out std_logic_vector (3 downto 0));
end entity;
```

### simulated with help of testbench\_PMconnect.vhd-

Here we divided testing into two parts for LDR and str instructions so,run each by uncommenting their portion half.For comparison among these 8 types of variations(varying minstr signal) we have kept mout and rout to same complex value-X"80049003".For selecting different bytes,word using adr signal.

We have introduced minstype signal to track currently which of 8 types of variations is running. It is 3 bit signal corresponding to minstr\_type: (STR,STRH,STRB,LDR,LDRH,LDRSH,LDRB,LDRSB,OTHR);

**test\_pmconnect\_str** -here all three types of str instruction and their variations are tested.



**test\_pmconnect\_ldr** -here all types of ldr instruction and their variations are tested



Note: To revert to EPWave opening in a new browser window, set that option on your user page.

### Merging stage 6 in main multicycle processor

PMconnect merge require appropriate adjustment 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).Also used state signal in testbench to show at which state running currently.

At this stage our design have only 9 states ,here we only added ONE new states –

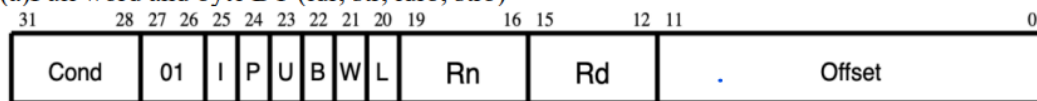
WB2RF-write back(same RES of ALU arithmetic ) to base resister in DT case when Wback = ‘1’ or Post-indexing.

here need to indrodue new state to access RF again .

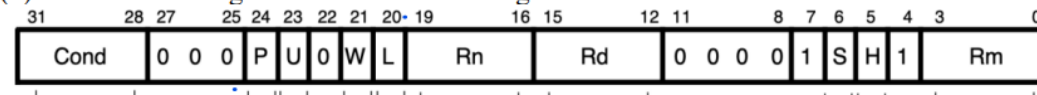
tried hardly less possible state diagram,given next page-

### **Format of Half word and Signed Data Transfer Instructions in ARM**

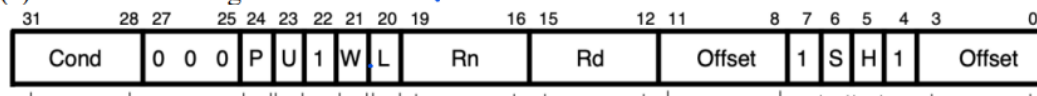
(a)Full word and byte DT (ldr, str, ldrb, strb)



(b) Half word and signed DT with offset in register



(c) Half word and signed DT with immediate offset



P, U, W and L bits have same meaning as in (a).A can be also register variant. Interpretation of S and H bits is as follows.

S H

- |   |   |   |
|---|---|---|
| 0 | 0 | swap (swp) - - not to be implemented            |
| 0 | 1 | unsigned half word load or store (ldrh or strh) |
| 1 | 0 | signed byte transfer load (ldrsb)               |
| 1 | 1 | signed half word load (ldrsh)                   |

### Summary of addressing modes-

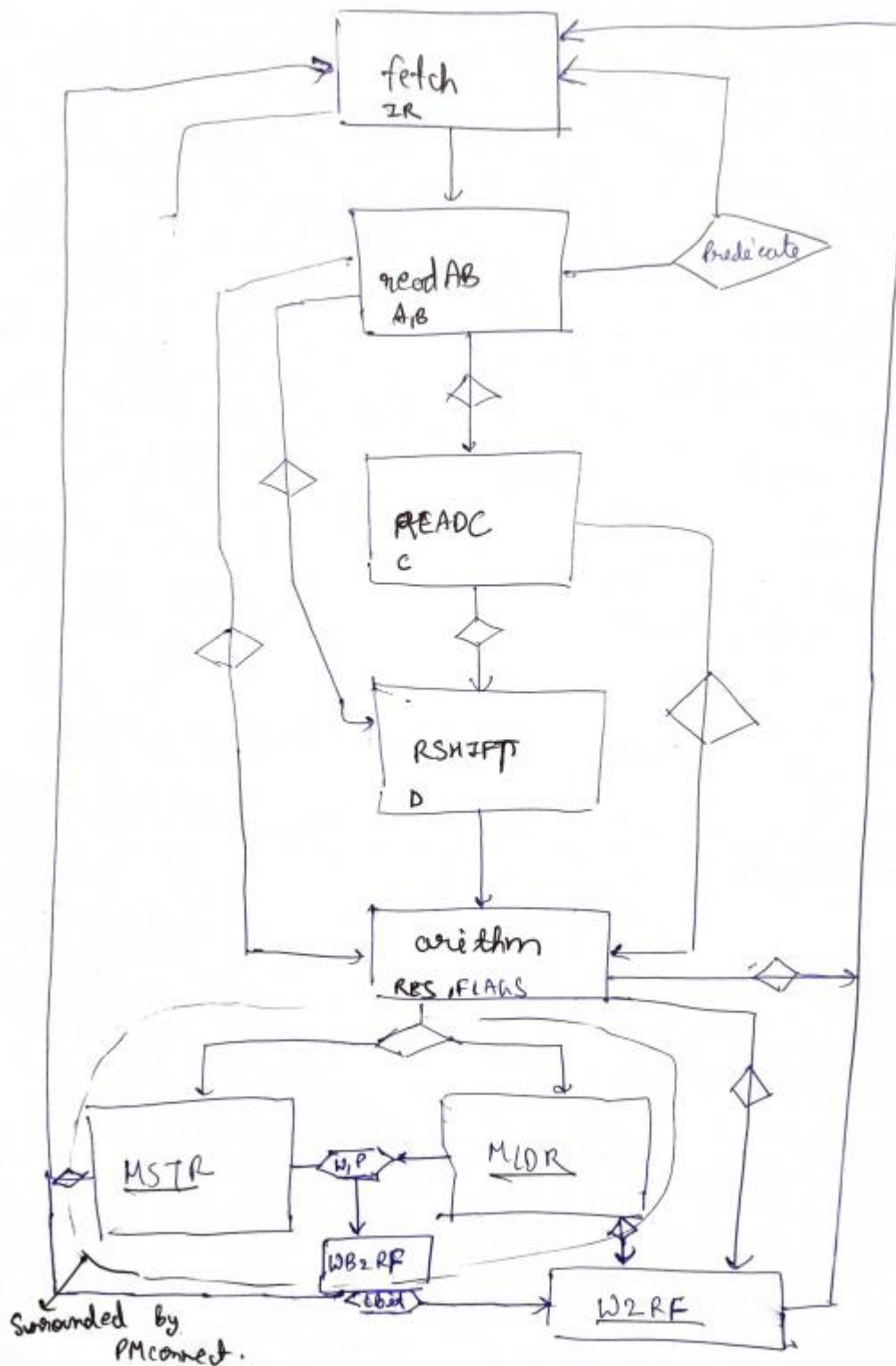
Syntax: <LDR|STR>{<cond>}{B} Rd, addressing<sup>1</sup>  
 LDR{<cond>}SB|H|SH Rd, addressing<sup>2</sup>  
 STR{<cond>}H Rd, addressing<sup>2</sup>

Addressing <sup>1</sup> mode and index method	Addressing <sup>1</sup> syntax
Preindex with immediate offset	[Rn, #+/-offset_12]
Preindex with register offset	[Rn, +/-Rm]
Preindex with scaled register offset	[Rn, +/-Rm, shift #shift_imm]
Preindex writeback with immediate offset	[Rn, #+/-offset_12]!
Preindex writeback with register offset	[Rn, +/-Rm]!
Preindex writeback with scaled register offset	[Rn, +/-Rm, shift #shift_imm]!
Immediate postindexed	[Rn], #+/-offset_12
Register postindex	[Rn], +/-Rm
Scaled register postindex	[Rn], +/-Rm, shift #shift_imm

### Practical idea used in implementing them in multicycle processor.

	Instruction	<i>r0</i> =	<i>r1</i> +=
Preindex with writeback	LDR r0, [r1, #0x4]!	mem32[r1+0x4]	0x4
	LDR r0, [r1, r2]!	mem32[r1+r2]	r2
	LDR r0, [r1, r2, LSR#0x4]!	mem32[r1+(r2 LSR 0x4)]	(r2 LSR 0x4)
Preindex	LDR r0, [r1, #0x4]	mem32[r1+0x4]	<i>not updated</i>
	LDR r0, [r1, r2]	mem32[r1+r2]	<i>not updated</i>
	LDR r0, [r1, -r2, LSR #0x4]	mem32[r1-(r2 LSR 0x4)]	<i>not updated</i>
Postindex	LDR r0, [r1], #0x4	mem32[r1]	0x4
	LDR r0, [r1], r2	mem32[r1]	r2
	LDR r0, [r1], r2, LSR #0x4	mem32[r1]	(r2 LSR 0x4)

## ASM CHART.



# 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.

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

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

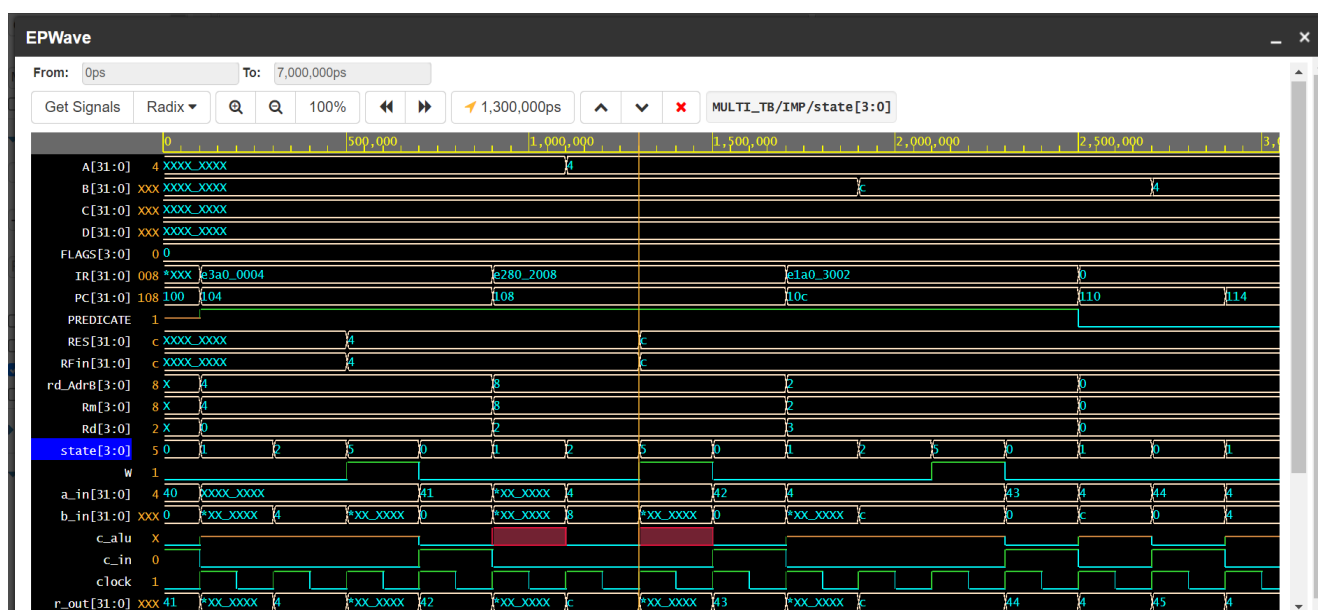
**Below are DT exhaustive related testcases-which covers all variants of indexing and transfer size instruction possible in DT.**

We can identify PMconnect related parameters in middle-end -MEMAd(address to memeory),MEMin, MEMout,rin, minstype,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).

**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 shifter stage Stestcase1-

RegistersView

General Purpose

Floating Point

Hexadecimal

Unsigned Decimal

Signed Decimal

R0 : 00000004  
R1 : 00000002  
R2 : 0000000c  
R3 : 00000000  
R4 : 0000000c  
R5 : 00000000  
R6 : 00000000  
R7 : 00000000  
R8 : 00000000  
R9 : 00000000  
R10 (s1) : 00000000  
R11 (fp) : 00000000  
R12 (ip) : 00000000  
R13 (sp) : 00011400  
R14 (lr) : 00000000  
R15 (pc) : 00011400

CPSR Register

Negative (N) : 0

Zero (Z) : 0

Carry (C) : 0

Overflow (V) : 0

CodeView

stestcase1.o

```

.text
00001000:E3A00004    mov r0,#4
00001004:E3A01002    mov r1,#2
00001008:E0802101    add r2,r0,r1,LSL #2
0000100C:E1A04002    mov r4,r2
.end...

```

OutputView

WatchView

Console

stdin/stdout/stderr



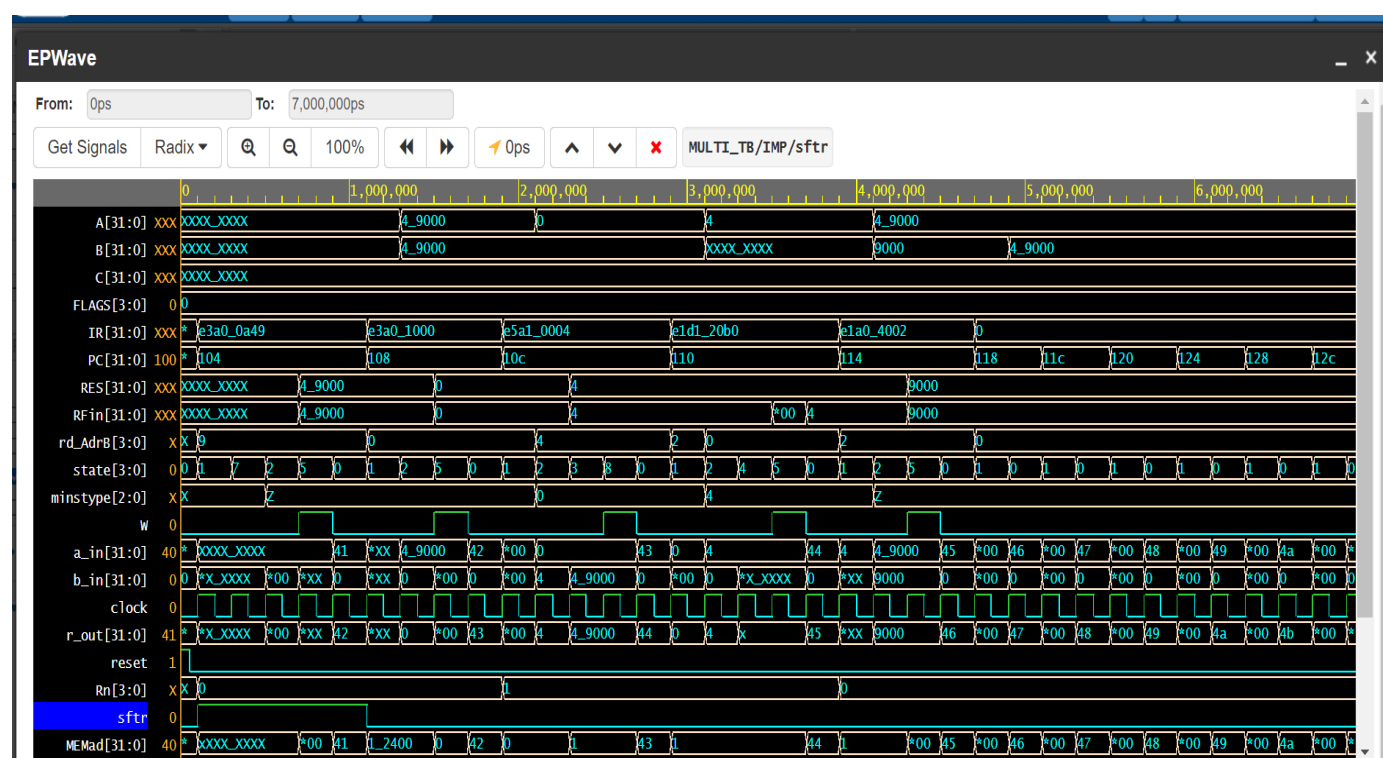
CodeView

pctestcase1.o

```

.text
00001000:E3A00A49    mov r0, #0X00049000
00001004:E3A01000    mov r1, #0
00001008:E5A10004    str r0, [r1, #4]!
0000100C:E1D120B0    ldrh r2, [r1]
00001010:E1A04002    mov r4, r2
.end

```



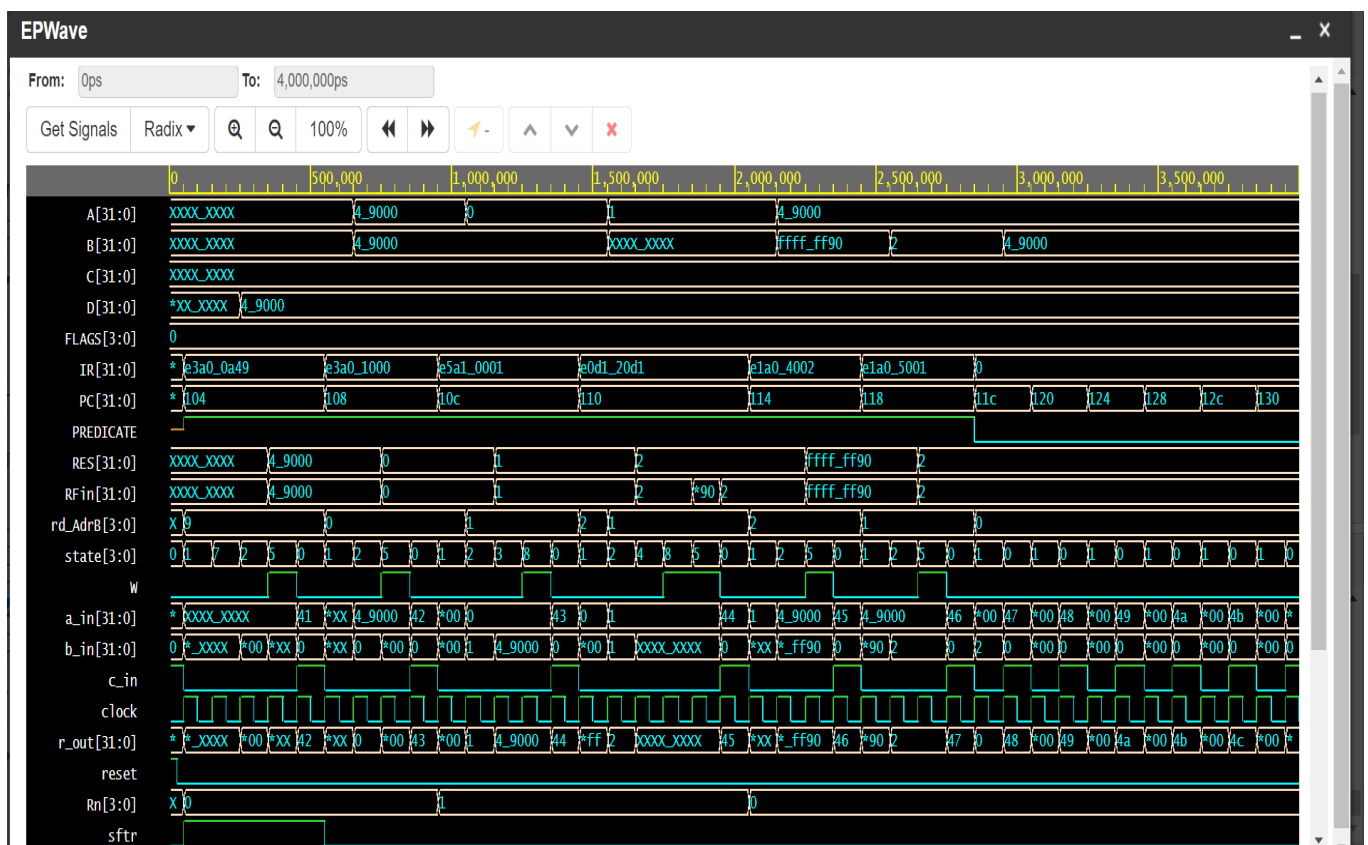
## PMtestcase2-

```

CodeView
pmtestcase2.o

.text
00001000:E3A00A49    mov r0, #0X00049000
00001004:E3A01000    mov r1,#0
00001008:E5A10001    str r0,[r1, #1]!
0000100C:E0D120D1    ldrsb r2,[r1],#1
00001010:E1A04002    mov r4,r2
00001014:E1A05001    mov r5,r1
.end

```

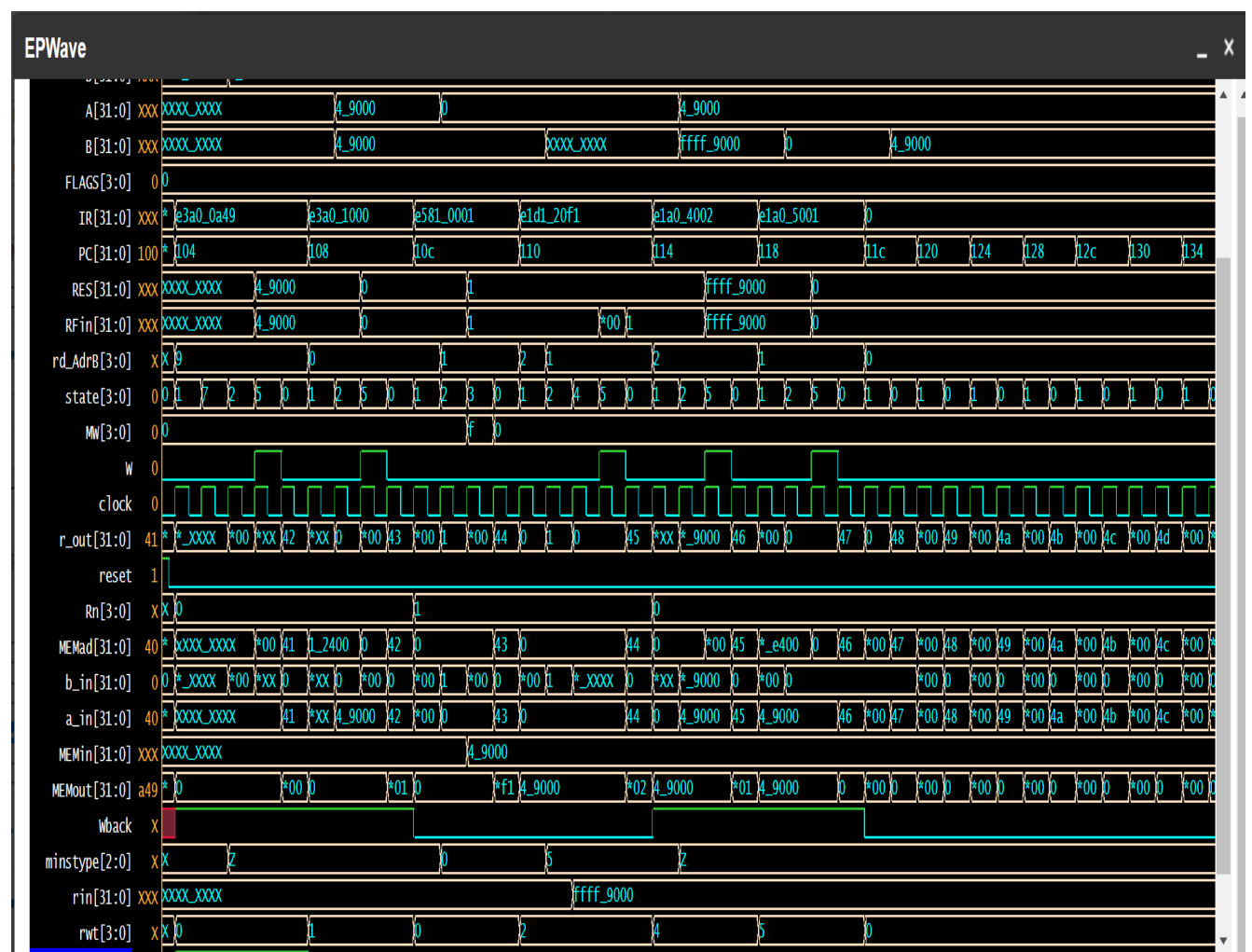


## PMtestcasE3.s

**CodeView**

pctestcase3.o

```
.text  
00001000:E3A00A49    mov r0, #0X00049000  
00001004:E3A01000    mov r1,#0  
00001008:E5810001    str r0,[r1, #1]  
0000100C:E1D120F1    ldrsh r2,[r1,#1]  
00001010:E1A04002    mov r4,r2  
00001014:E1A05001    mov r5,r1  
.end
```



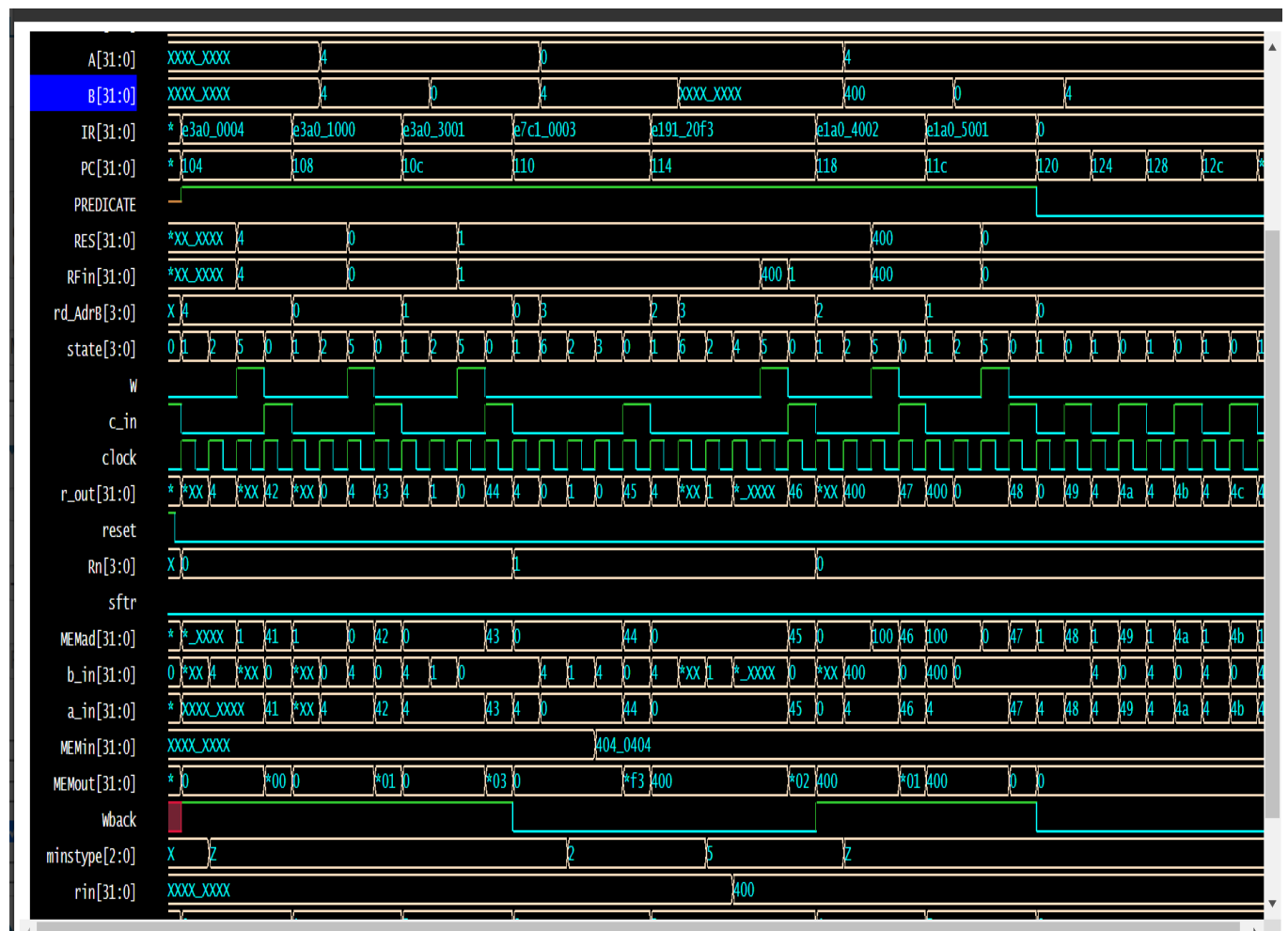
# PMtestcasE4.s

```

CodeView
pmtestcase4.o

.text
00001000:E3A00004    mov r0, #0X00000004
00001004:E3A01000    mov r1,#0
00001008:E3A03001    mov r3,#1
0000100C:E7C10003    strb r0,[r1, r3]
00001010:E19120F3    ldrsh r2,[r1,r3]
00001014:E1A04002    mov r4,r2
00001018:E1A05001    mov r5,r1
.end

```



## PMtestcasE5.s

X

CodeView

pctestcase5.o

.text

00001000:E3A00801   mov r0, #0X00010000

00001004:E3A01000   mov r1,#0

00001008:E3A03001   mov r3,#1

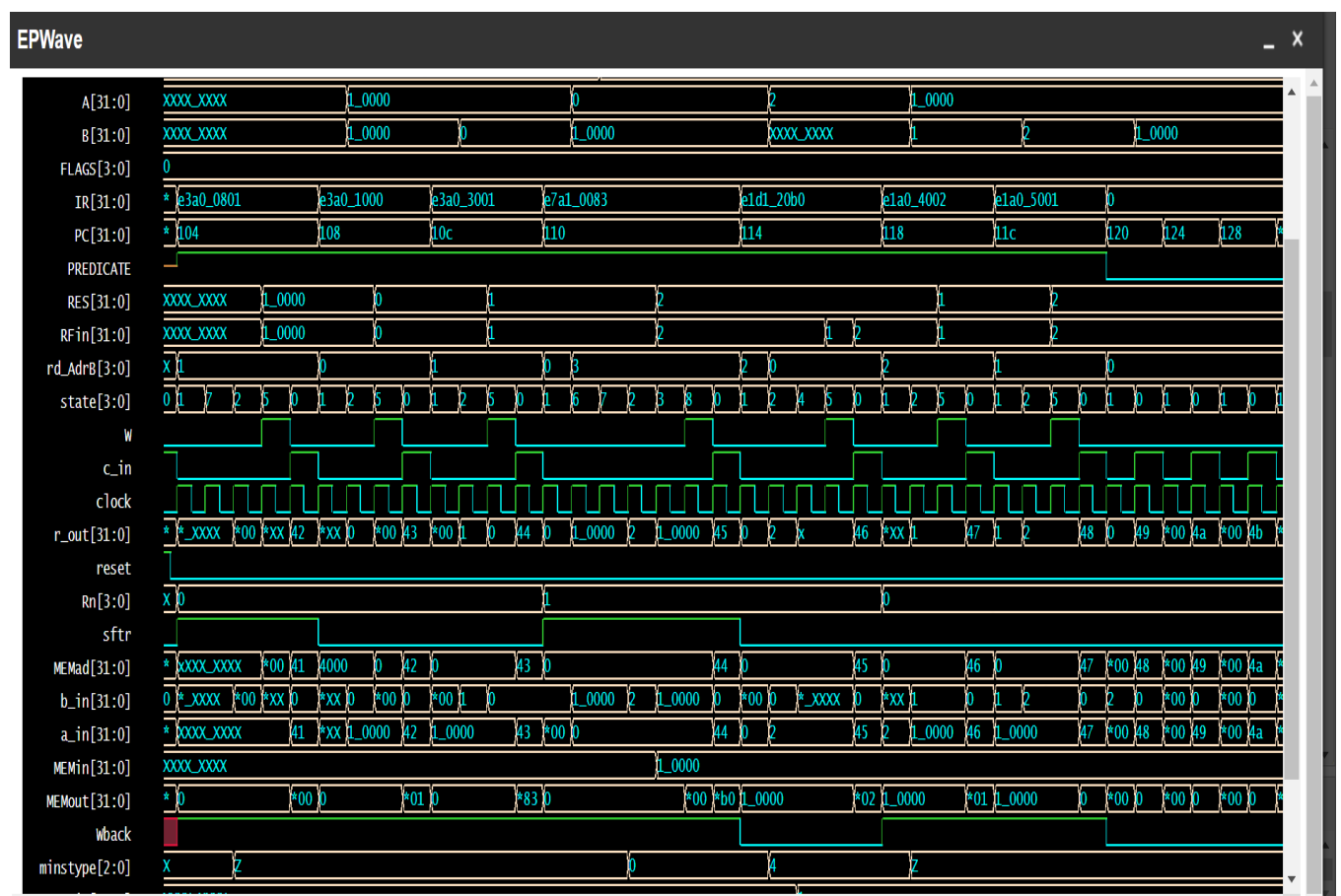
0000100C:E7A10083   str r0,[r1,r3,LSL #1]!

00001010:E1D120B0   ldrh r2,[r1]

00001014:E1A04002   mov r4,r2

00001018:E1A05001   mov r5,r1

.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-

The screenshot displays the edaplayground.com web interface. The top navigation bar includes a logo, a 'Run' button, and a 'Save' button. The main workspace is divided into three panels. The left panel, titled 'Languages & Libraries', contains a 'Testbench + Design' section with a 'VHDL' dropdown, a 'Libraries' section with a 'None' dropdown, and a 'Tools & Simulators' section with 'Aldec Riviera Pro 2020.04' selected. The middle panel shows a VHDL testbench file named 'testbench.vhd' with lines 314 to 343. The right panel shows a VHDL design file named 'PM.vhd' with lines 56 to 82. The bottom panel shows a log window with the message '[2022-03-22 21:35:25 UTC]. Opening EPWave...' and a 'Done' button. An 'EPWave' window is also visible in the bottom right corner.

```
314 WAIT FOR 50 ns;
315 CLK<='0';
316 WAIT FOR 50 ns;
317
318 CLK<='1';
319 WAIT FOR 50 ns;
320 CLK<='0';
321 WAIT FOR 50 ns;
322 CLK<='1';
323 WAIT FOR 50 ns;
324 CLK<='0';
325 WAIT FOR 50 ns;
326 CLK<='1';
327 WAIT FOR 50 ns;
328 CLK<='0';
329 WAIT FOR 50 ns;
330 CLK<='1';
331 WAIT FOR 50 ns;
332 CLK<='0';
333 WAIT FOR 50 ns;
334 CLK<='1';
335 WAIT FOR 50 ns;
336 CLK<='0';
337 WAIT FOR 50 ns;
338 CLK<='1';
339 WAIT FOR 50 ns;
340 CLK<='0';
341 WAIT FOR 50 ns;
342
343 ASSERT FALSE REPORT "Test done. Open EPWave to see"
```

```
56 -- others => X"00000000"
57 -- );
58
59 --PMtestcase 4--1
60 -- SIGNAL MEMORY : DMEM:=
61 -- (64 => X"E3A00004",
62 -- 65 => X"E3A01000",
63 -- 66 => X"E3A03001",
64 -- 67 => X"E7C10003",
65 -- 68 => X"E19120F3",
66 -- 69 => X"E1A04002",
67 -- 70 => X"E1A05001",
68 -- others => X"00000000"
69 -- );
70 --PMtestcase 5--1
71 SIGNAL MEMORY : DMEM:=
72 (64 => X"E3A00801",
73 65 => X"E3A01000",
74 66 => X"E3A03001",
75 67 => X"E7A10083",
76 68 => X"E1D120B0",
77 69 => X"E1A04002",
78 70 => X"E1A05001",
79 others => X"00000000"
80 );
81 -- --stestcase 1--1
82 -- SIGNAL MEMORY : DMEM:=
```

[2022-03-22 21:35:25 UTC]. Opening EPWave...

Done

EPWave

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