Computer Architecture Experiment

Topic 4. Pipelined CPU with Cache

浙江大学计算机学院

Outline

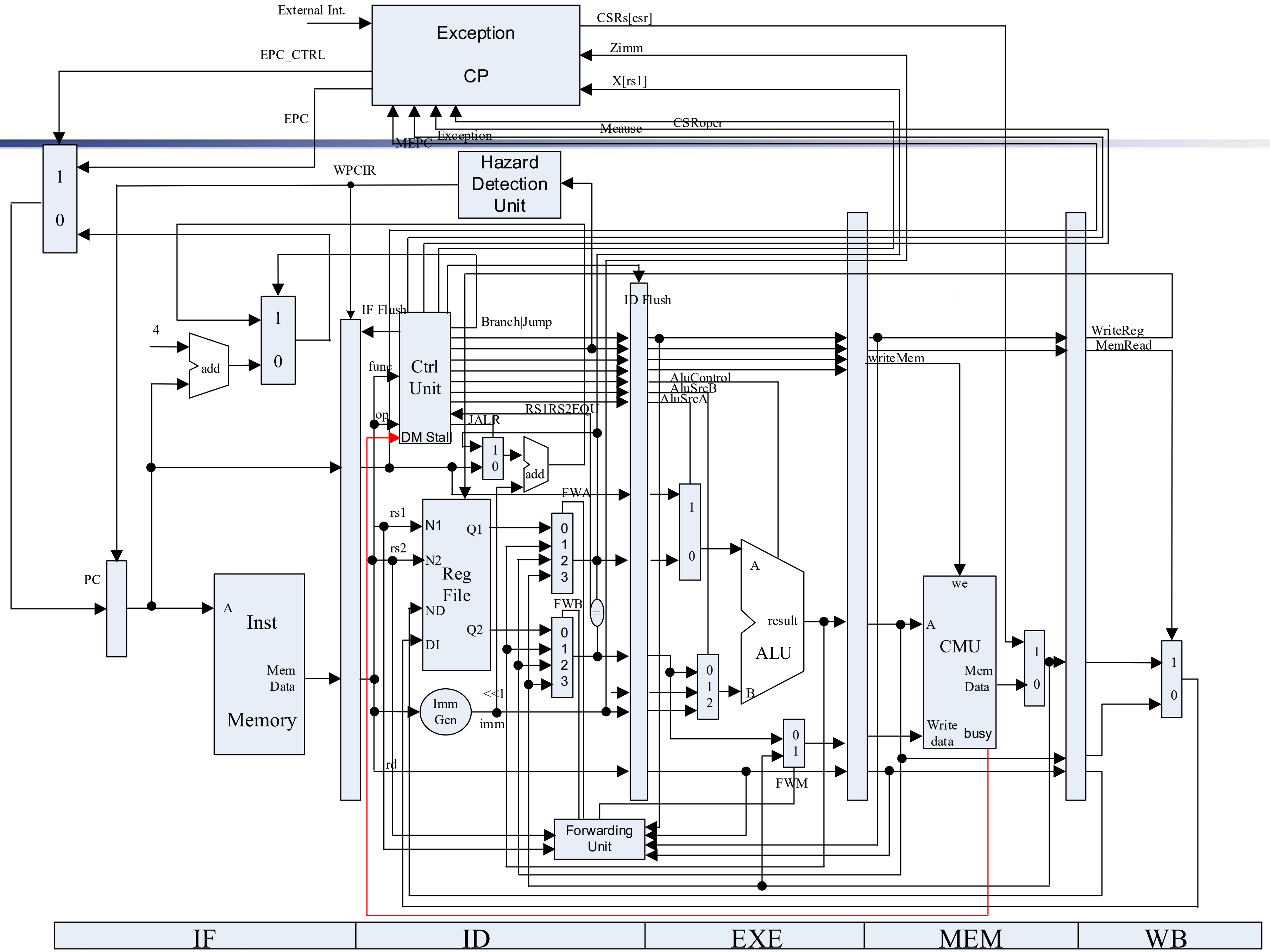
- Experiment Purpose
- Experiment Task
- Basic Principle
- Operating Procedures
- Checkpoints

Experiment Purpose

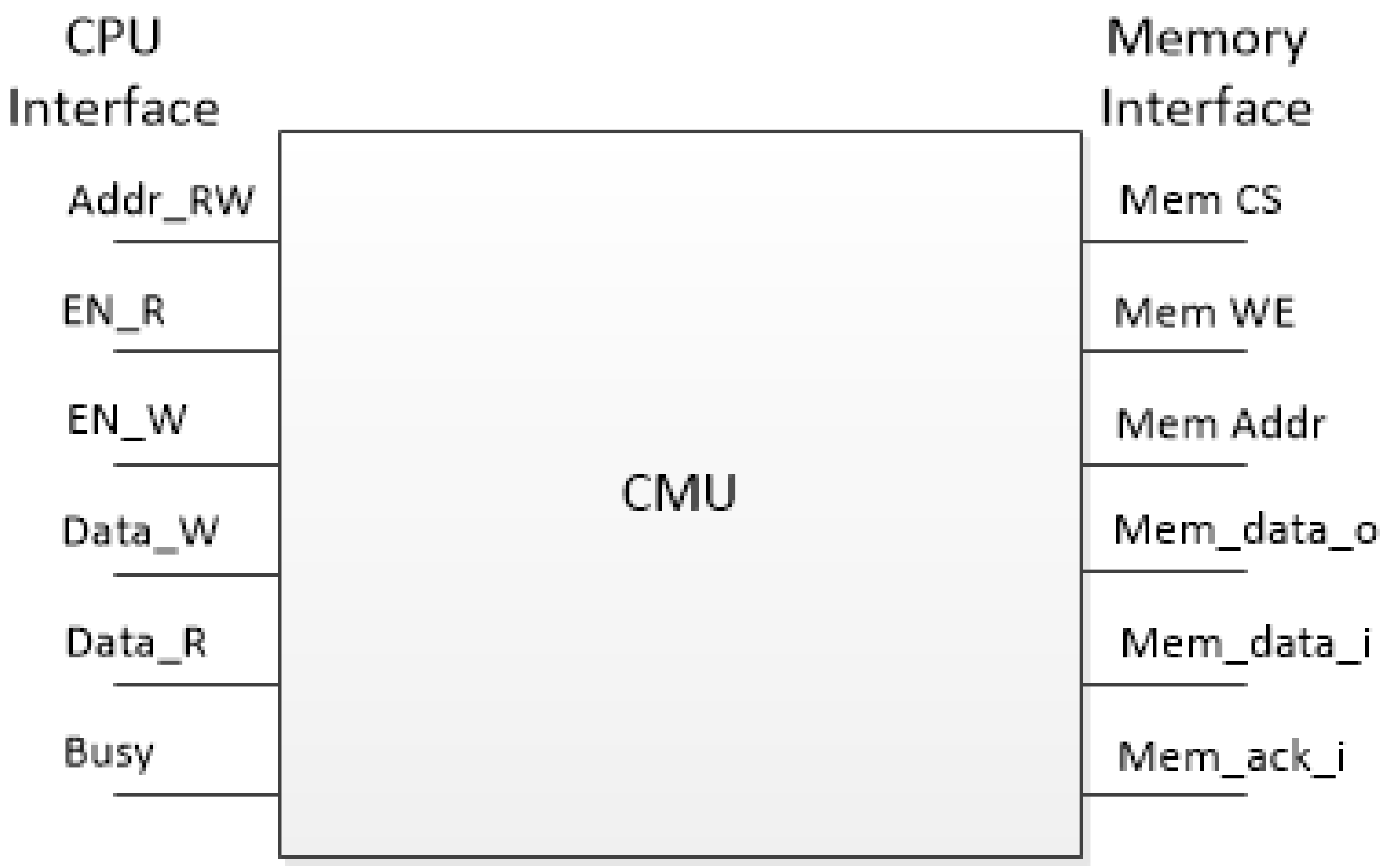
- **Understand the principle of Cache Management Unit (CMU) and State Machine of CMU**
- **Master the design methods of CMU and Integrate it to the CPU.**
- **Master verification methods of CMU and compare the performance of CPU when it has cache or not.**

Experiment Task

- Design of Cache Management Unit and integrate it to CPU.
- Observe and Analyze the Waveform of Simulation.
- Compare the performance of CPU when it has cache or not.

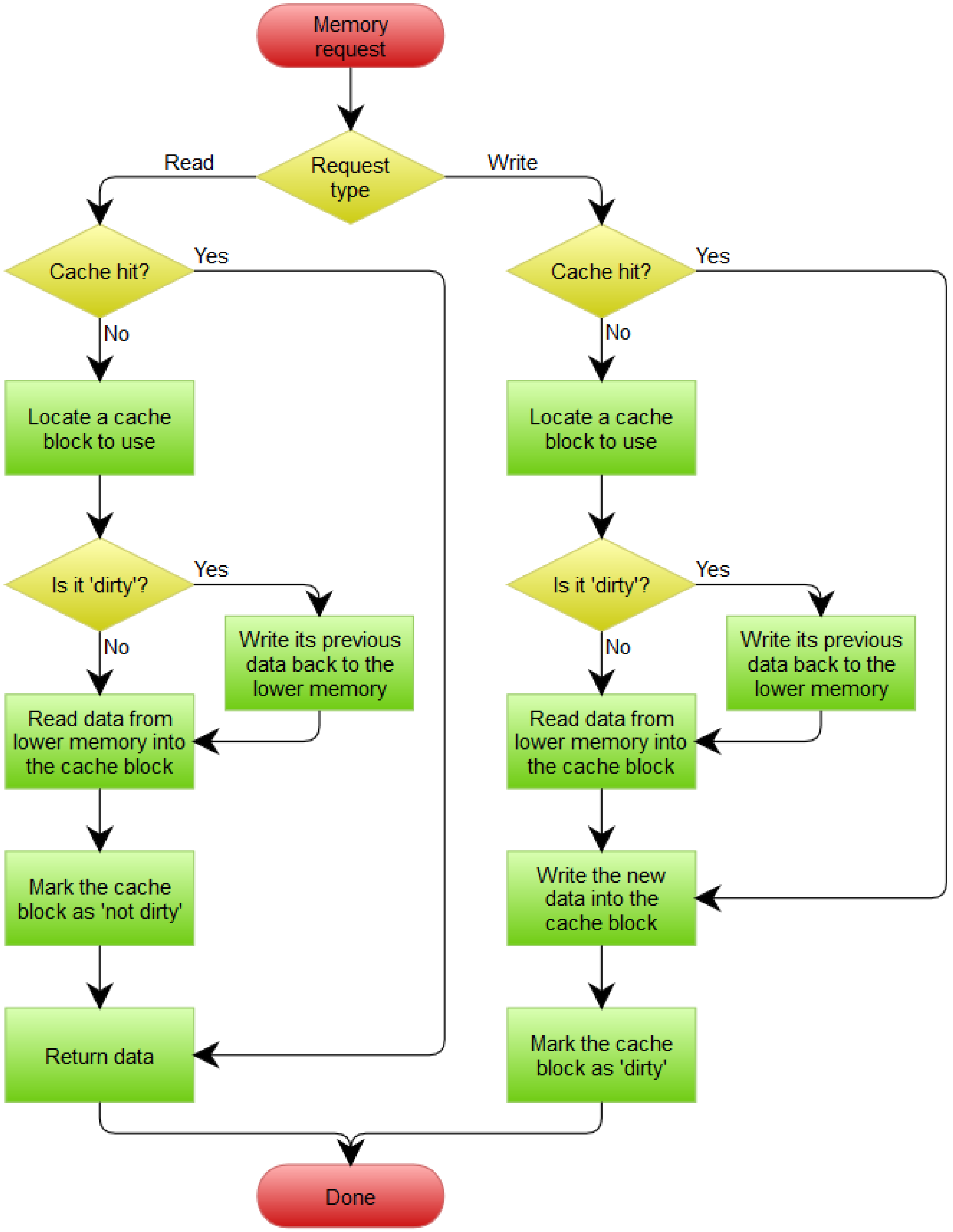

External Int.
Exception
CP
CSRs[csr]
Zimm
X[rs1]
EPC_CTRL
EPC
MEPC
Exception
Meause
CSRoper
WPCIR
Hazard Detection Unit
IF Flush
ID Flush
Branch|Jump
func
Ctrl Unit
op
DM Stall
RS1RS2EQU
JALR
AluControl
AluSrcB
AluSrcA
writeMem
WriteReg
MemRead
4
add
PC
Inst Memory
A
Mem Data
rs1
rs2
N1
N2
ND
DI
Q1
Q2
Reg File
FWA
FWB
FWM
Imm Gen
imm
<<1
rd
ALU
result
B
CMU
we
Write data
busy
Forwarding Unit
IF
ID
EXE
MEM
WB

Cache Management Unit

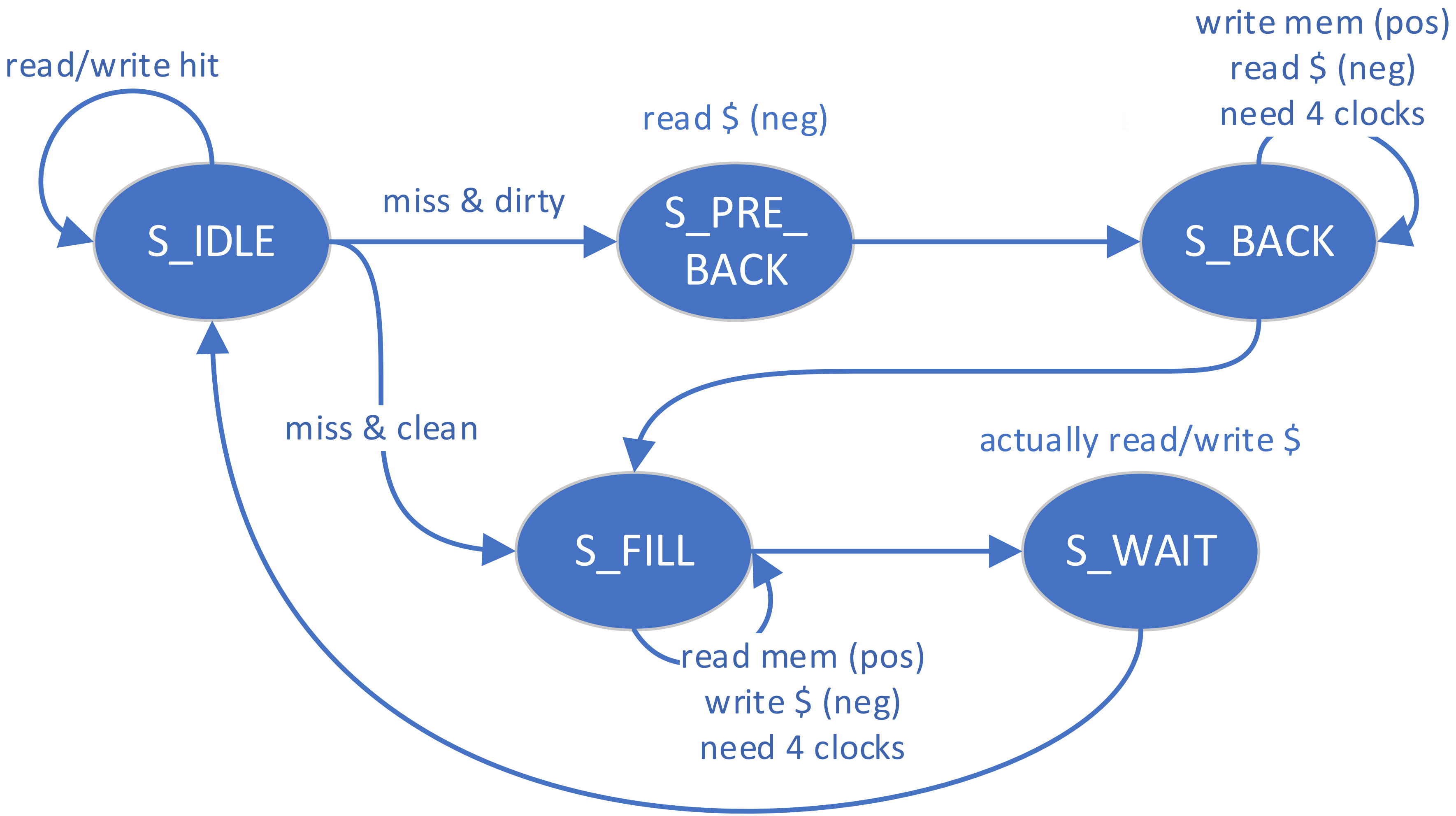


Cache Operation Flc

- **Read (Hit/Miss)**
- **Write (Hit/Miss)**
- **Replace (Clean/Dirty)**

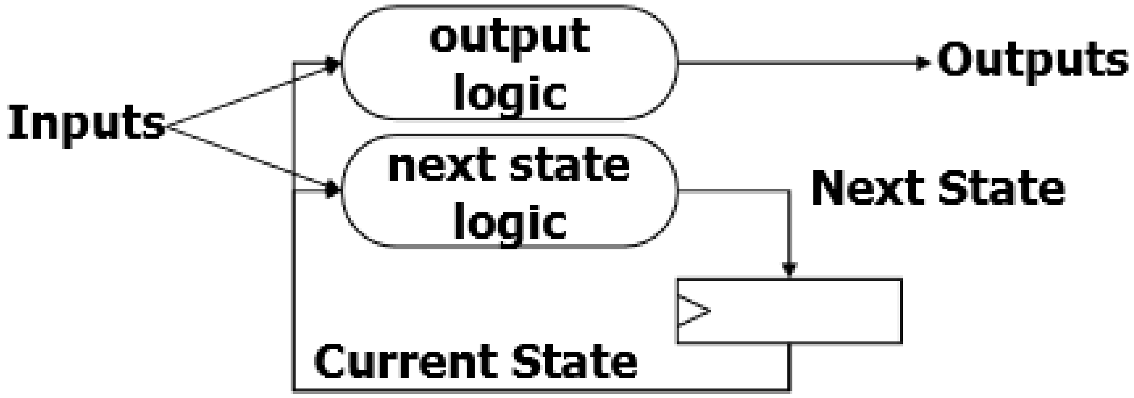


Cache Management State Machine



State Machine

- Next State Logic
- State assignment
- Output



Next logic (1)

```
S_IDLE: begin
    if (en_r || en_w) begin
        if (cache_hit)
            next_state = ???;
        else if (cache_valid && cache_dirty)
            next_state = ???;
        else
            next_state = ???;
    end
    next_word_count = 2'b00;
end

S_PRE_BACK: begin
    next_state = ???;
    next_word_count = 2'b00;
end
```

Next logic (2)

```
S_BACK: begin
    if (mem_ack_i && word_count == {ELEMENT_WORDS_WIDTH{1'b1}})
// 2'b11 in default case
        next_state = ???;
    else
        next_state = ???;

    if (mem_ack_i)
        next_word_count = ???;
    else
        next_word_count = word_count;
 end
```

Next logic (3)

```
S_FILL: begin
    if (mem_ack_i && word_count == {ELEMENT_WORDS_WIDTH{1'b1}})
        next_state = ???;
    else
        next_state = ???;

    if (mem_ack_i)
        next_word_count = ???;
    else
        next_word_count = word_count;
end

S_WAIT: begin
    next_state = ???;
    next_word_count = 2'b00;
end
```

Perform State Assignment

```
always @ (posedge clk) begin
    if (rst) begin
        state <= S_IDLE;
        word_count <= 2'b00;
    end
    else begin
        state <= next_state;
        word_count <= next_word_count;
    end
end
```

Output (1)

```
case(state)
        S_IDLE, S_WAIT: begin
           cache_addr = addr_rw;
           cache_load = en_r;
           cache_edit = en_w;
           cache_store = 1'b0;
           cache_u_b_h_w = u_b_h_w;
           cache_din = data_w;
        end
        S_BACK, S_PRE_BACK: begin
           cache_addr = {addr_rw[ADDR_BITS-1:BLOCK_WIDTH], next_word_count, {ELEMENT_WORDS_WIDTH{1'b0}}};
           cache_load = 1'b0;
           cache_edit = 1'b0;
           cache_store = 1'b0;
           cache_u_b_h_w = 3'b010;
           cache_din = 32'b0;
        end
        S_FILL: begin
           cache_addr = {addr_rw[ADDR_BITS-1:BLOCK_WIDTH], word_count, {ELEMENT_WORDS_WIDTH{1'b0}}};
           cache_load = 1'b0;
           cache_edit = 1'b0;
           cache_store = mem_ack_i;
           cache_u_b_h_w = 3'b010;
           cache_din = mem_data_i;
        end
     endcase
```

Output (2)

```
case (next_state)
        S_IDLE, S_PRE_BACK, S_WAIT: begin
           mem_cs_o = 1'b0;
           mem_we_o = 1'b0;
           mem_addr_o = 32'b0;
        end

        S_BACK: begin
           mem_cs_o = 1'b1;
           mem_we_o = 1'b1;
           mem_addr_o = {cache_tag, addr_rw[ADDR_BITS-TAG_BITS-1:BLOCK_WIDTH], next_word_count, {ELEMENT_WORDS_WIDTH{1'b0}}};
        end

        S_FILL: begin
           mem_cs_o = 1'b1;
           mem_we_o = 1'b0;
           mem_addr_o = {addr_rw[ADDR_BITS-1:BLOCK_WIDTH], next_word_count, {ELEMENT_WORDS_WIDTH{1'b0}}};
        end
endcase
```

RV32 Core

```
module cmu (
        input clk,                          // CPU side
        input rst,
        input [31:0] addr_rw,
        input en_r,
        input en_w,
        input [2:0] u_b_h_w,
        input [31:0] data_w,
        output [31:0] data_r,
        output stall,
        output reg mem_cs_o = 0,            // mem side
        output reg mem_we_o = 0,
        output reg [31:0] mem_addr_o = 0,
        input [31:0] mem_data_i,
        output [31:0] mem_data_o,
        input mem_ack_i,
        output [2:0] cmu_state              // debug info
    );
```

Instr. Mem.(1)

NO.	Instruction	Addr.	Label	ASM	Comment
0	00000013	0	__start:	addi x0, x0, 0	
1	01c00083	4		lb x1, 0x01C(x0)	# F0F0F0F0 in 0x1C # FFFFFFF0 miss, read 0x010~0x01C to set 1 line 0
2	01c01103	8		lh x2, 0x01C(x0)	# FFFFF0F0 hit
3	01c02183	C		lw x3, 0x01C(x0)	# F0F0F0F0 hit
4	01c04203	10		lbu x4, 0x01C(x0)	# 000000F0 hit
5	01c05283	14		lhu x5, 0x01C(x0)	# 0000F0F0 hit
6	21002003	18		lw x0, 0x210(x0)	# miss, read 0x210~0x21C to cache set 1 line 1
7	abcde0b7	1C		lw x7, 20(x0)	
8	402200b3	20		lui x1 0xABCDE	
9	71c08093	24		addi x1, x1, 0x71C	# x1 = 0xABCDE71C
10	00100023	28		sb x1, 0x0(x0)	# miss, read 0x000~0x00C to cache set 0 line 0
11	00101223	2C		sh x1, 0x4(x0)	# hit
12	00102423	30		sw x1, 0x8(x0)	# hit

Instr. Mem.(2)

NO.	Instruction	Addr.	Label	ASM	Comment
13	20002303	34		lw x6, 0x200(x0)	# miss, read 0x200~0x20C to cache set 0 line 1
14	40002383	38		lw x7, 0x400(x0)	# miss, write 0x000~0x00C back to ram, then read 0x400~40C to cache set 0 line 0
15	41002403	3C		lw x8, 0x410(x0)	# miss, no write back because of clean, read 0x410~41C to chache set 1 line 0
16	0ed06813	40	loop:	lori x16, x0, 0xED	# end
17	ffdff06f	44		jal x0, loop	

Data Mem.

NO.	Data	Addr.	Comment
0	000080BF	0	
1	00000008	4	
2	00000010	8	
3	00000014	C	
4	FFFF0000	10	
5	0FFF0000	14	
6	FF000F0F	18	
7	F0F0F0F0	1C	
8	00000000	20	
9	00000000	24	
10	00000000	28	
11	00000000	2C	
12	00000000	30	
13	00000000	34	
14	00000000	38	
15	00000000	3C	

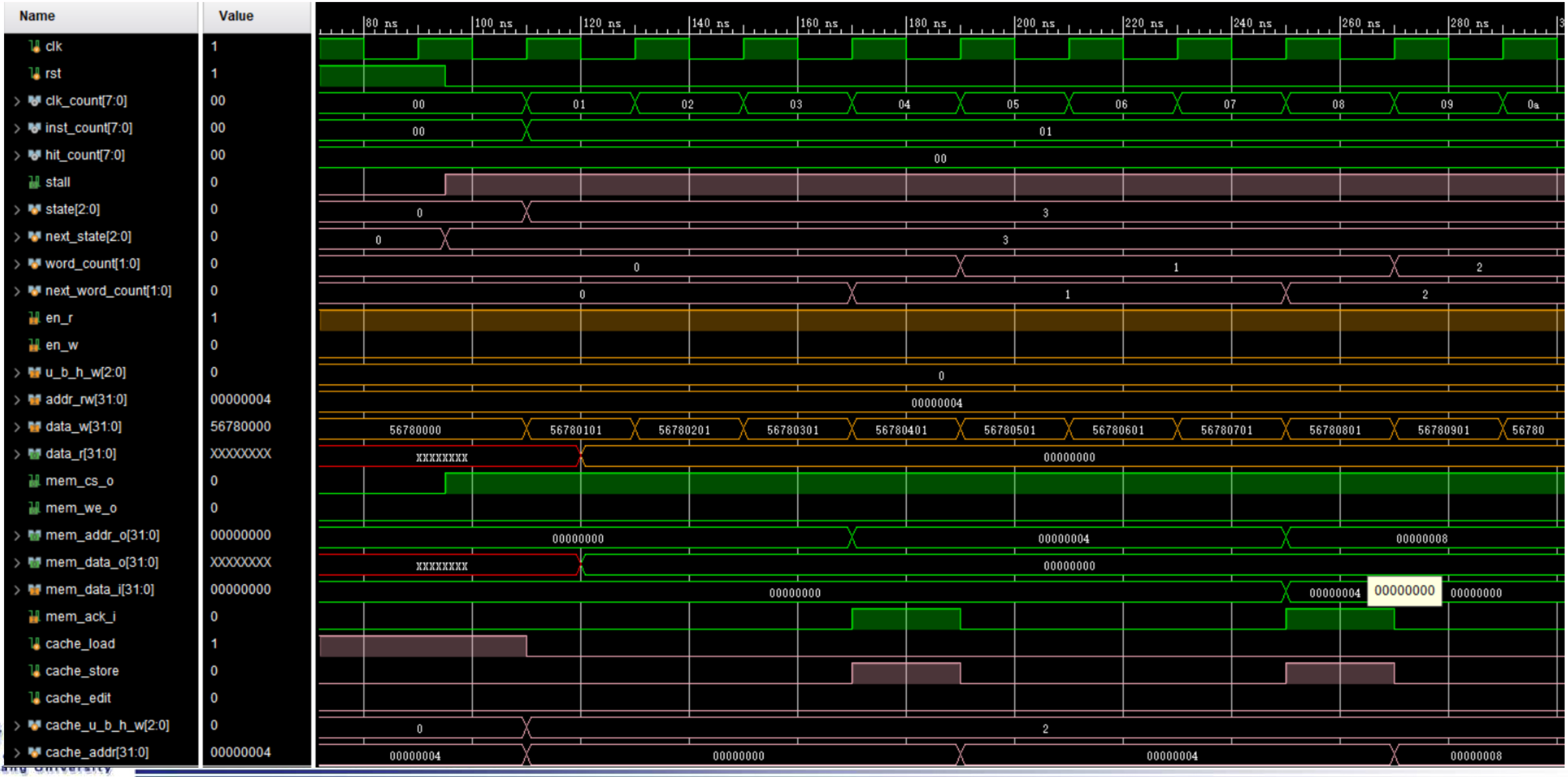
NO.	Instruction	Addr.	Comment
16	00000000	40	
17	00000000	44	
18	00000000	48	
19	00000000	4C	
20	A3000000	50	
21	27000000	54	
22	79000000	58	
23	15100000	5C	
24	00000000	60	
25	00000000	64	
26	00000000	68	
27	00000000	6C	
28	00000000	70	
29	00000000	74	
30	00000000	78	
31	00000000	7C	

Test Bench

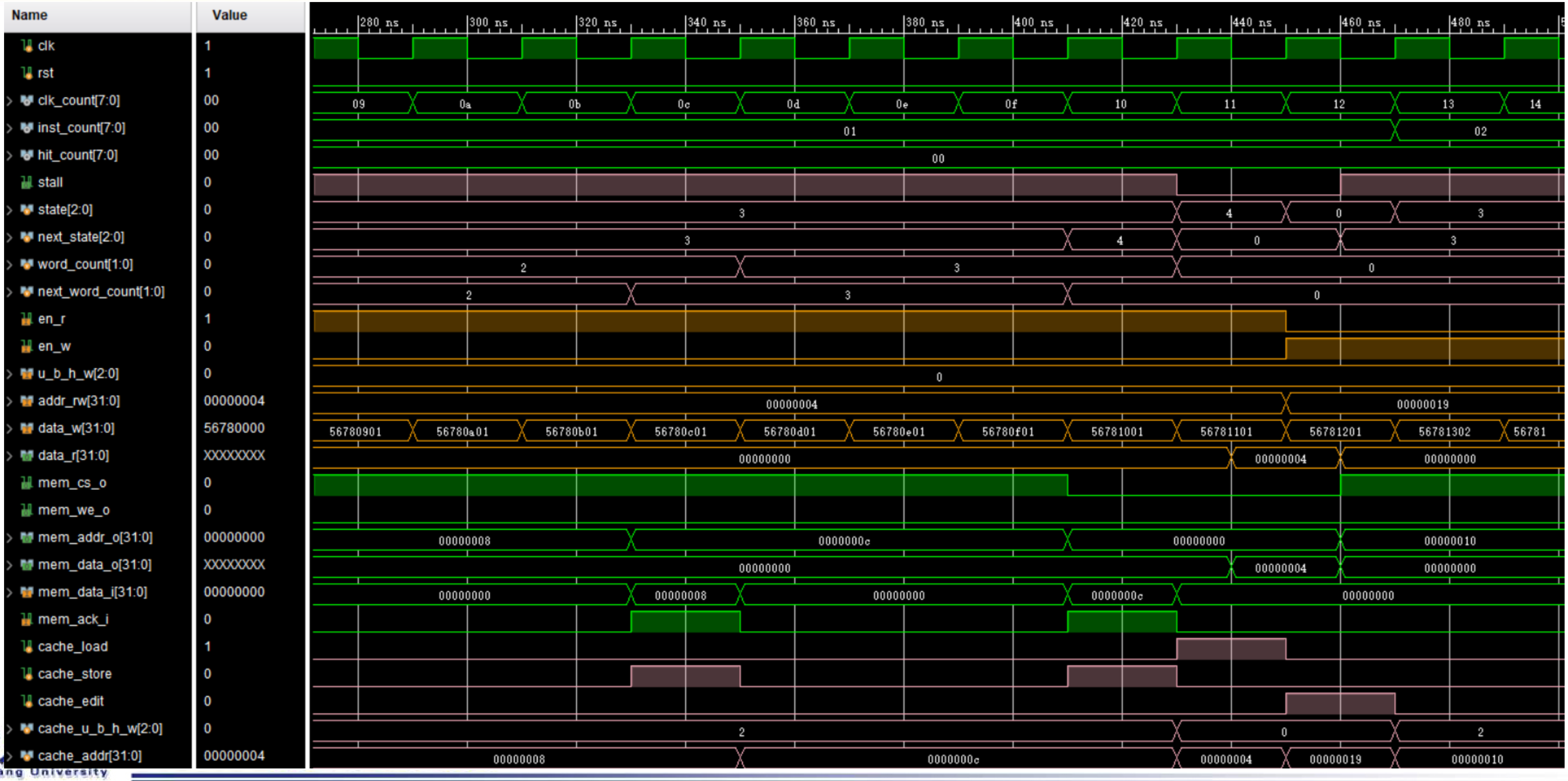
```
initial begin
        data[0] = 40'h0_2_00000004;  // read miss              1+17
        data[1] = 40'h0_3_00000019;  // write miss             1+17
        data[2] = 40'h1_2_00000008;  // read hit               1
        data[3] = 40'h1_3_00000014;  // write hit              1

        data[4] = 40'h2_2_00000204;  // read miss              1+17
        data[5] = 40'h2_3_00000218;  // write miss             1+17
        data[6] = 40'h0_3_00000208;  // write hit              1
        data[7] = 40'h4_2_00000414;  // read miss + dirty      1+17+17
        data[8] = 40'h1_3_00000404;  // write miss + clean     1+17
        data[9] = 40'h0;             // end                    total: 128
end
```

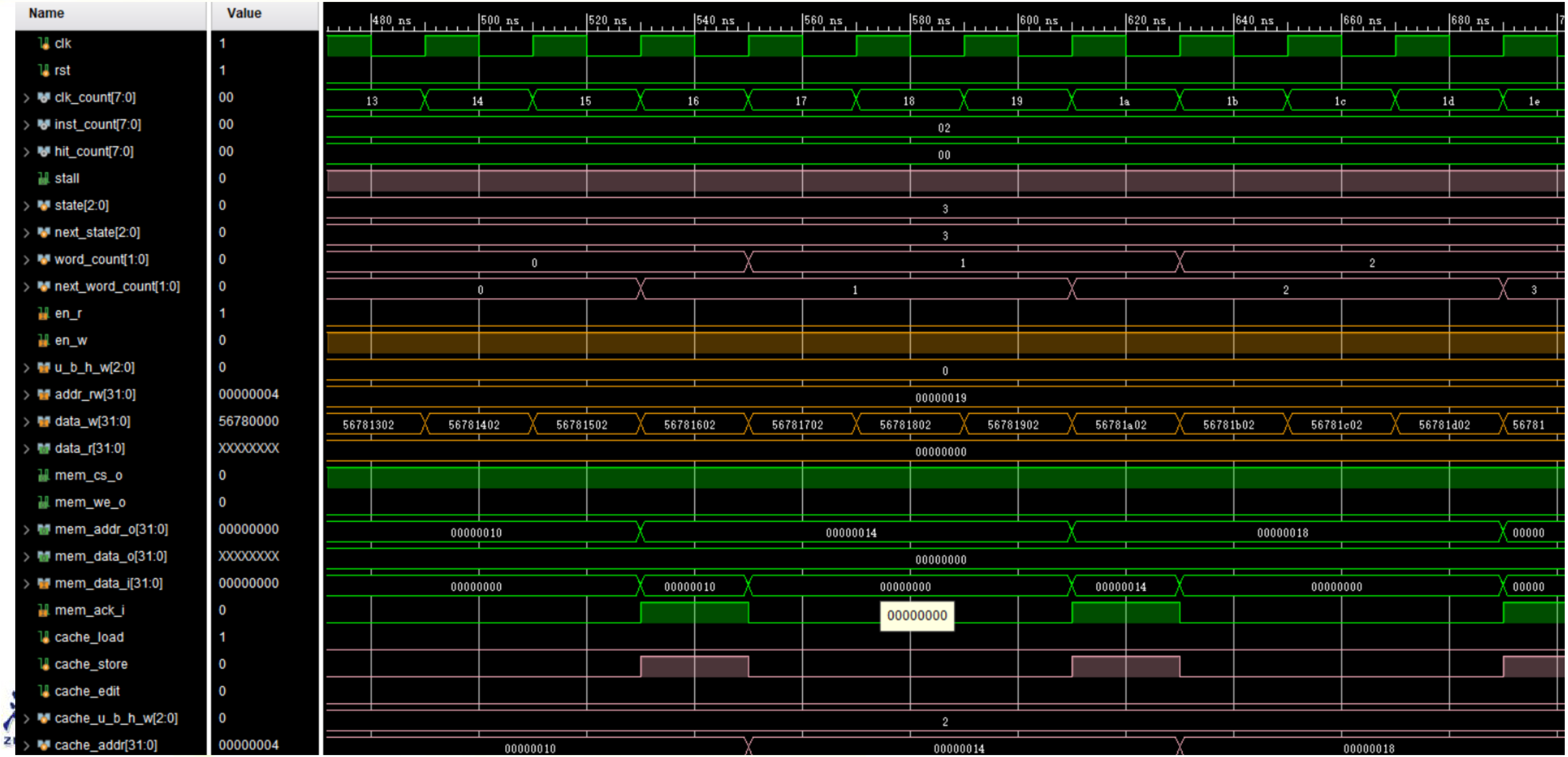
Simulation(1)



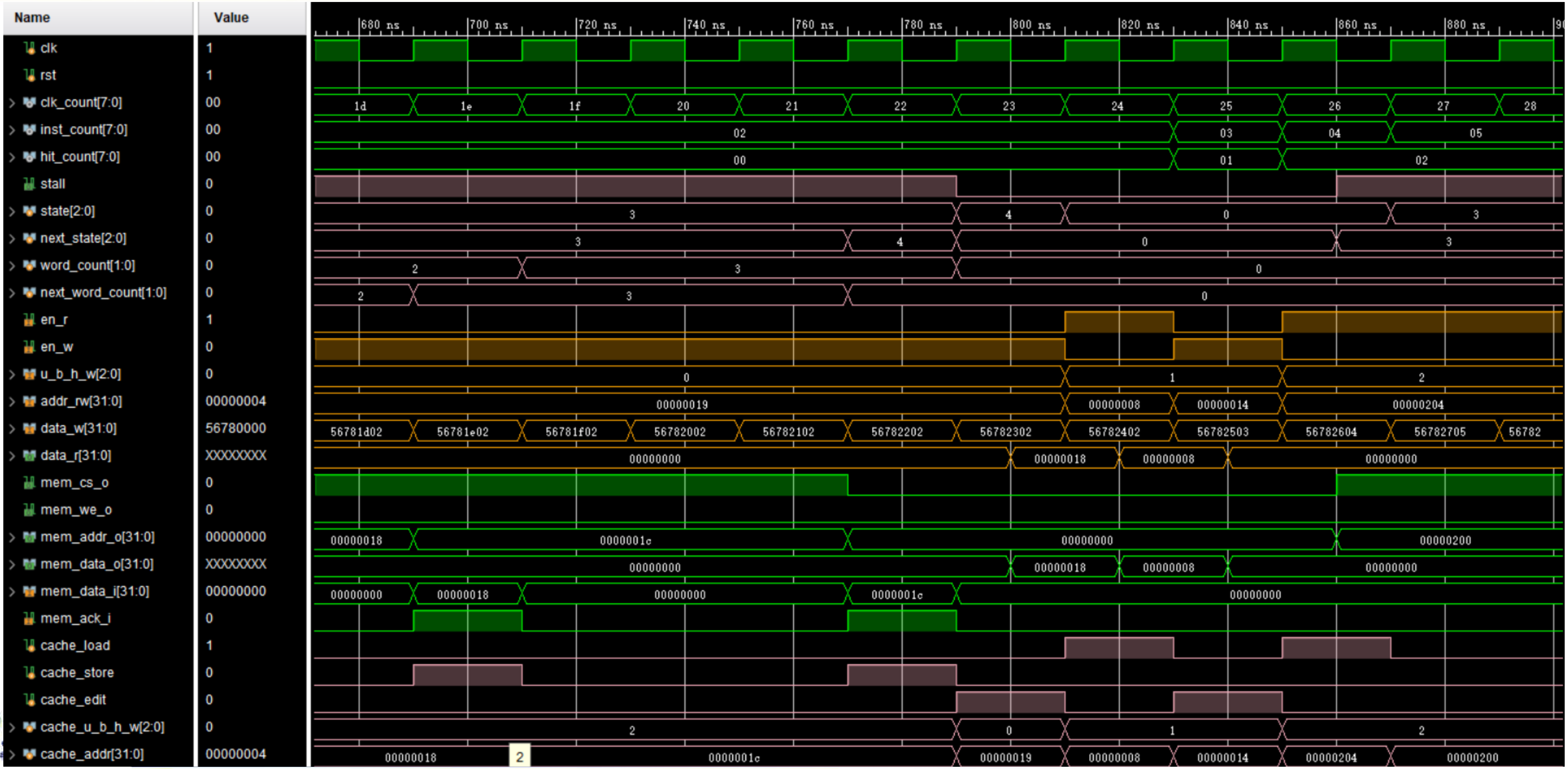
Simulation(2)



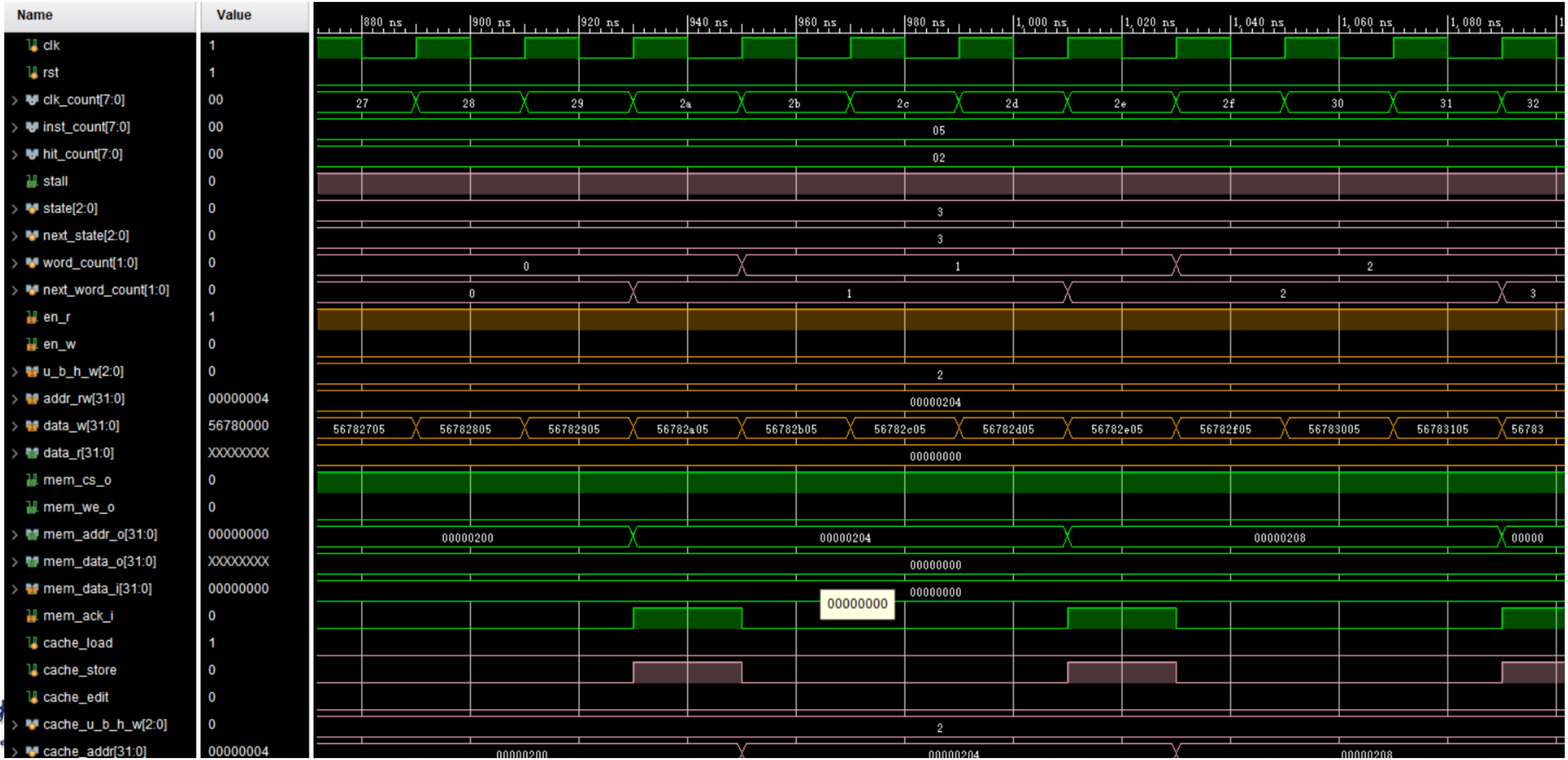
Simulation(3)



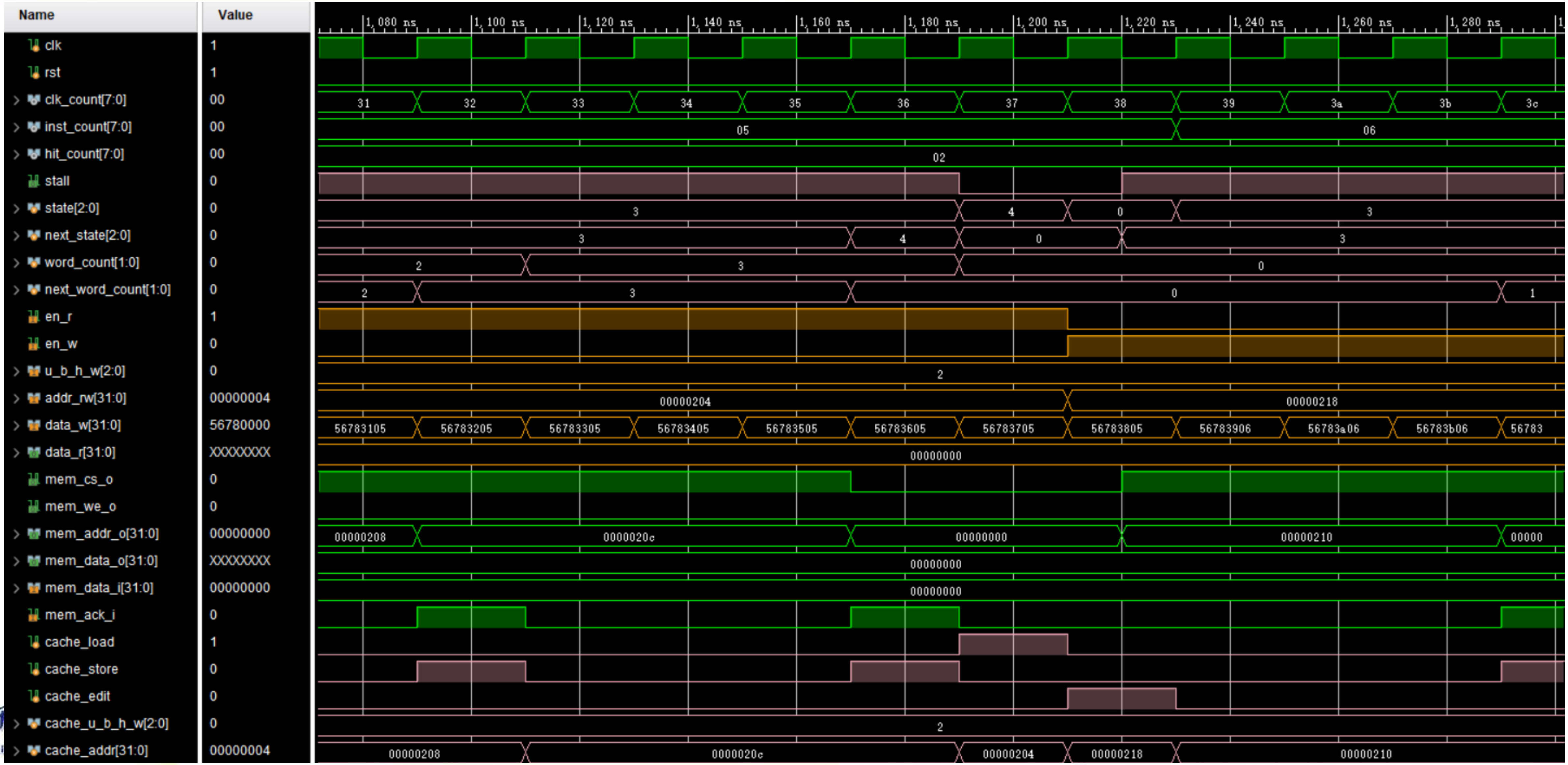
Simulation(4)



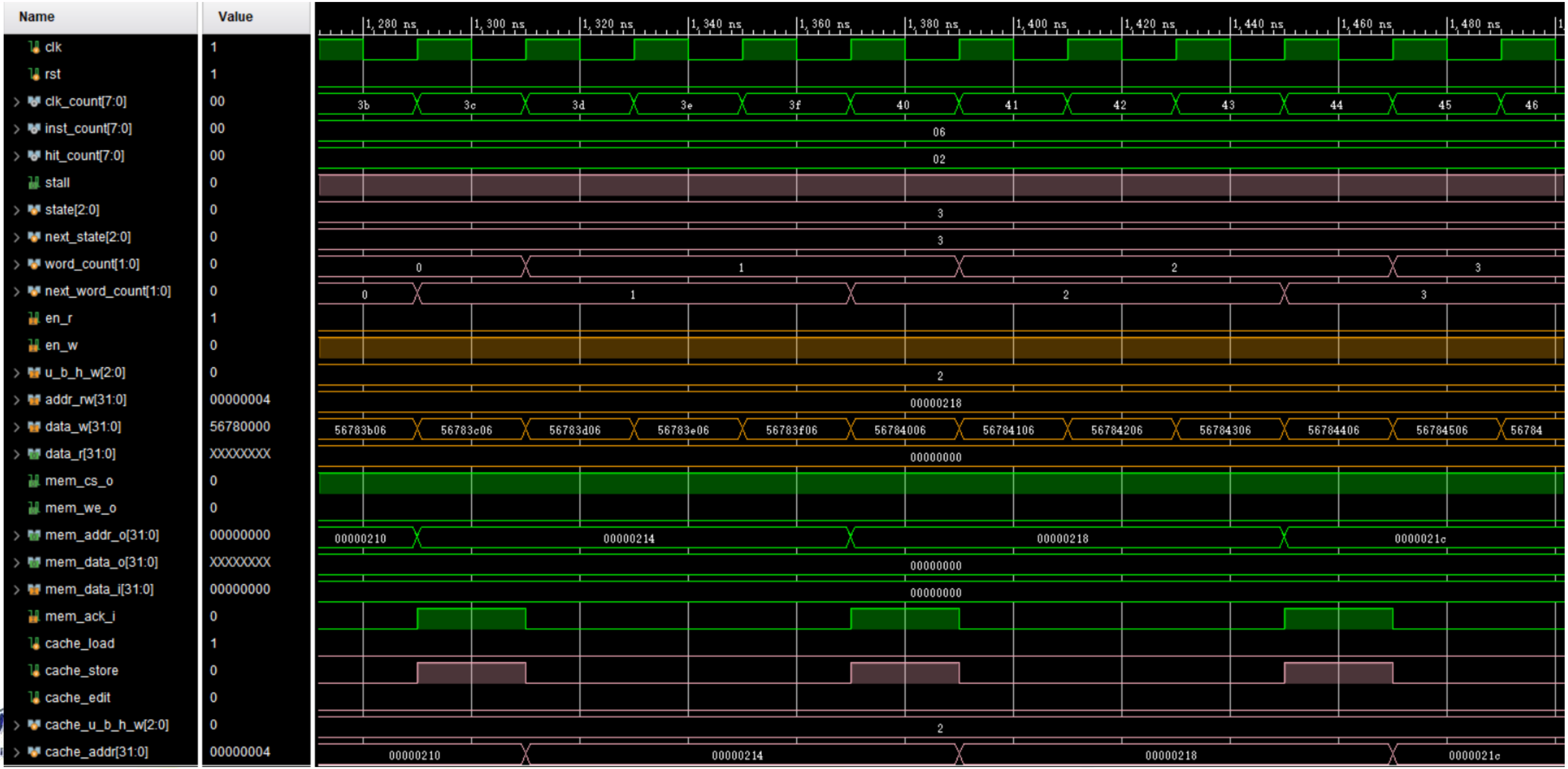
Simulation(5)



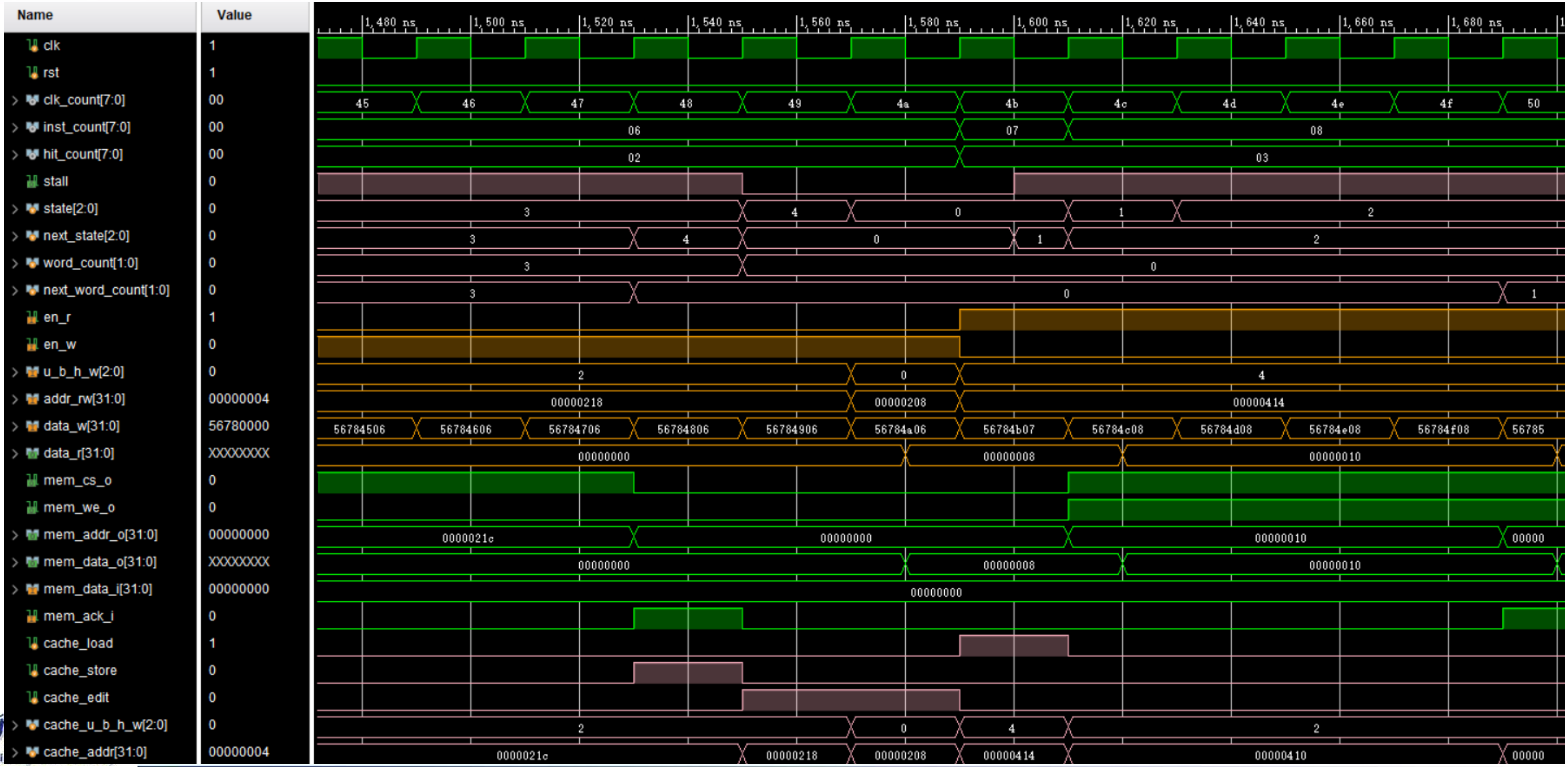
Simulation(6)



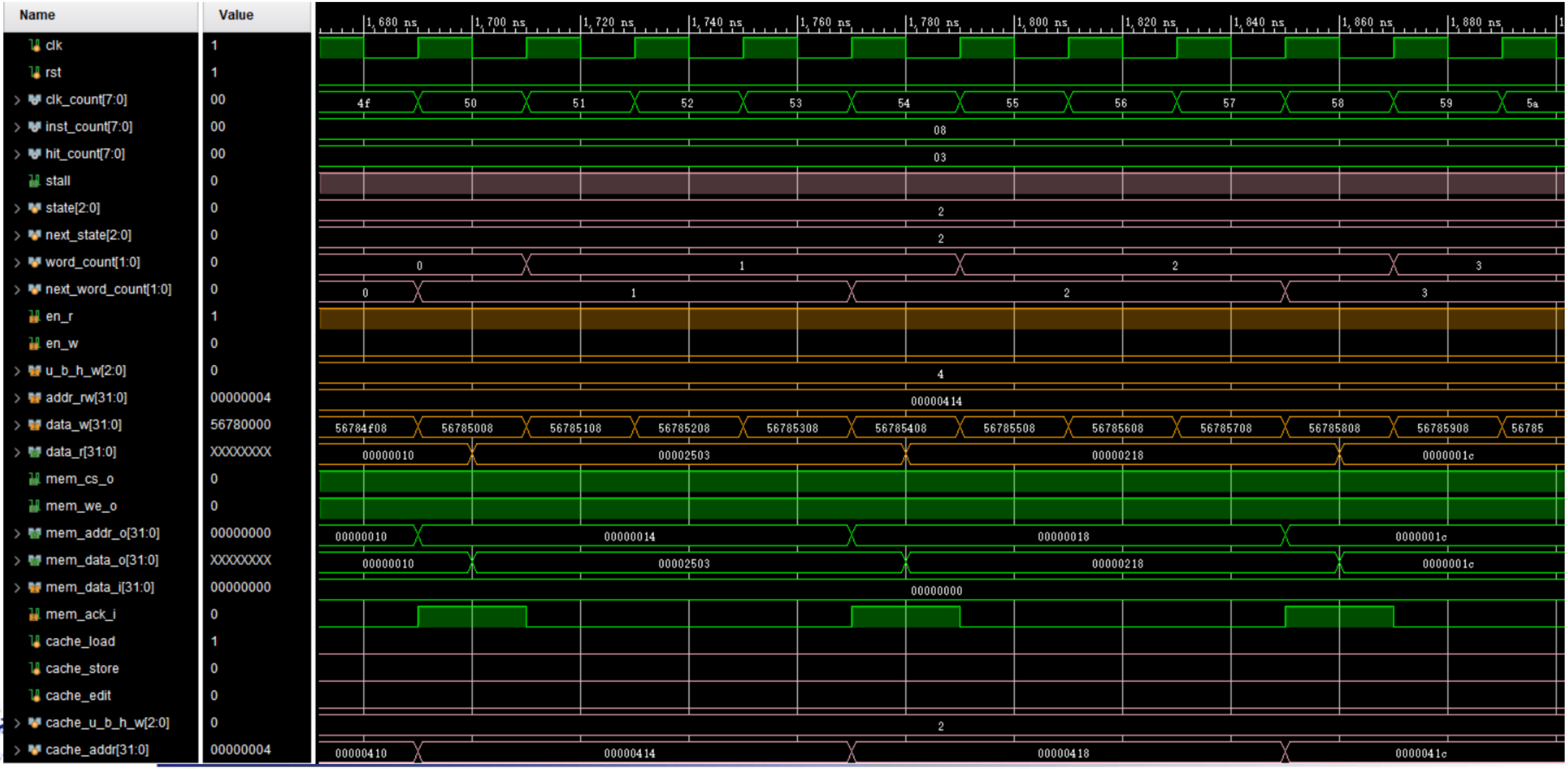
Simulation(7)



Simulation(8)



Simulation(9)



Checkpoints

- **CP1:**

  Waveform Simulation of CMU.

- **CP2:**

  FPGA Verification.

Thanks!