# 112-1 SoC Design Laboratory

# LabD

Group 10 王語卉 吳至淩 高宇勝

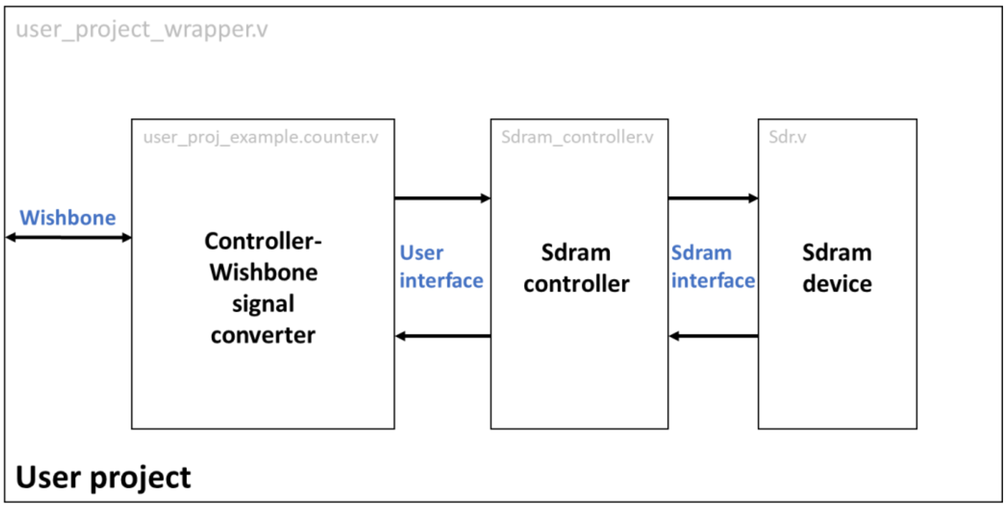
## Introduction

In this lab, we replace the BRAM with SDRAM to store the data and the firmware code. We've done three tasks in this lab: (1) replace the adder with matrix multiplier, (2) add the prefetch function in SDRAM controller, (3) separate data and firmware into two banks in SDRAM.

## Observations

### <u>SDRAM controller design and SDRAM bus protocol</u>

<u>Overall structure</u>



<u>User interface</u>

There 7 signals in user interface: user_addr, rw, data_in, data_out, busy, in_valid, out_valid. Users can use these signals to read or write to the sdram.

user_addr: the address to read/write.

rw: read or write, read is 0 and write is 1.

data_in: the data that needs to be written.

data_out: the data which is read out by sdram.

in_valid: the read/write request.

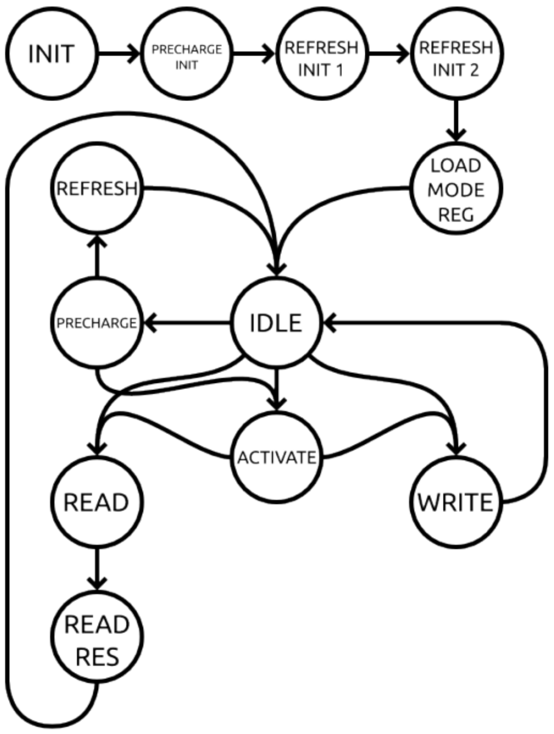
out_valid: pulse high when the data to be read out is prepared.

<u>Sdram interface</u>

The sdram interface contain: sdram_cle, sdram_cs, sdram_cas, sdram_ras, sdram_we, sdram_dqm, sdram_ba, sdram_a (address), sdram_dqi (input data), sdram_dqo (output data). The interface is controlled by the sdram controller.

<u>Sdram controller</u>

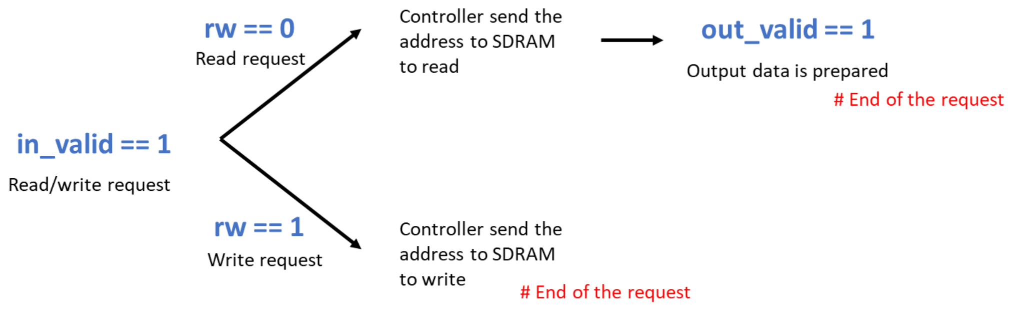
Sdram controller is controlled by the user interface, and the controller will send the address to sdram to fetch or write the data. There's an FSM in the controller:



If the bank that need to be accessed has not been activated, then we need to **activate** before read/write. If we need to change the bank to access, then we need to **precharge** this bank first, **activate** the target bank, then we can do a read/write. If after 750T, all the bank needs to be **precharged** and **refresh**.

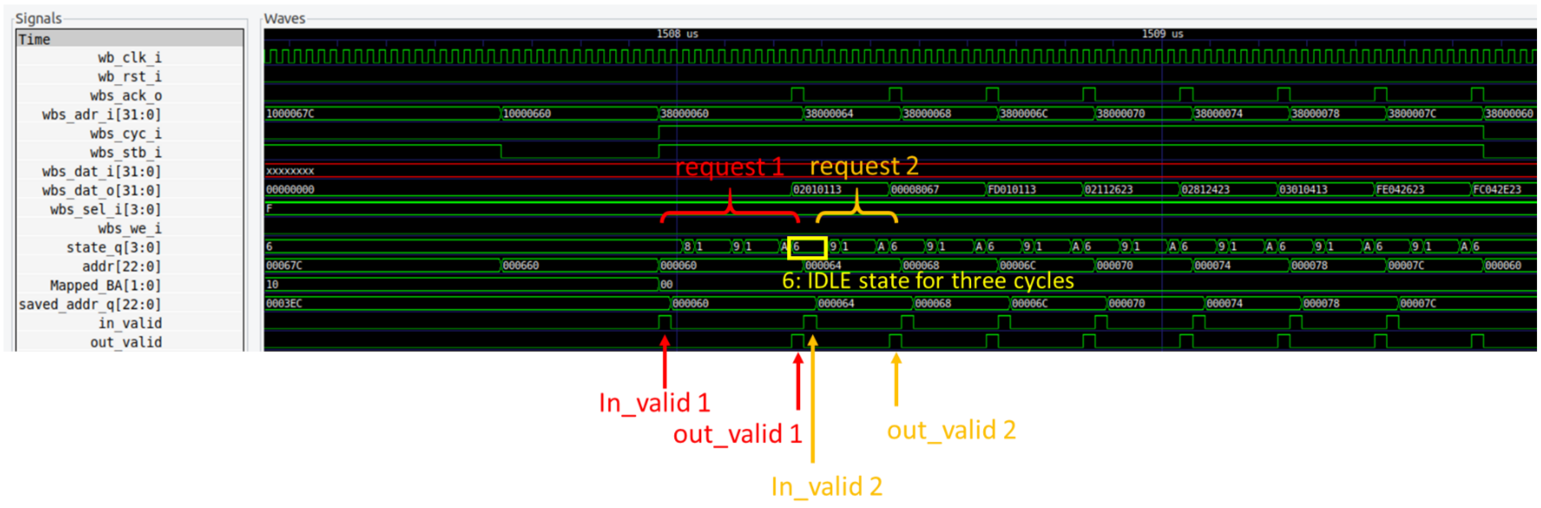
The operation flow in the sdram controller is like below:

## Operation flow



## The prefetch scheme

Since the controller will stay in IDLE state for three cycles after sending out the previous data without receiving the next request. We want to reduce the time the controller stays in IDLE state by sending the probable next address to the controller. If the next address matches the address we sent, then we can save two redundant cycles. If not, then we design a scheme to block out the output valid signal.



Our scheme: after request 1, directly load (address 1 + 4) into SDRAM without receiving the next address (address 2). When in_valid goes high for request 2, we compare (address 1 + 4) with address 2 to check if our preloaded address is correct. If is not correct (miss), then we block out the out_valid signal to prevent sending out the wrong data.

The signals we use:

**Prefetch**: 1 bit flag, goes high three cycles after in_valid == 1, goes 0 when there's no input request or output is not valid. If the prefetch is 1, then address + 4 will be load into SDRAM for the next read.

```
prefetch_n = (~in_req | out_not) ? 1'b0 : in_valid2 ? 1'b1 : prefetch;
```

```
if (ready_q && (in_valid | in_valid3)) begin
    saved_rw_d = rw;
    saved_data_d = data_in;
    saved_addr_d = addr + {prefetch, 2'b0};
    ready_d = 1'b0;
end
```

New address which will send to SDRAM

**in_req**: monitor of input read request

```
assign read_req = in_valid & ~rw;

in_req_n = out_valid ? 1'b0 : read_req ? 1'b1 : in_req;
```

**out_not**: output is not valid. Will be triggered if loaded address is not equal to the new address.

```
out_not_n = out_valid_q ? 1'b0 : (in_valid & addr_flag & (addr != saved_addr_q)) ? 1'b1 : out_not;

assign out_valid = out_valid_q & !out_not & in_req;
```

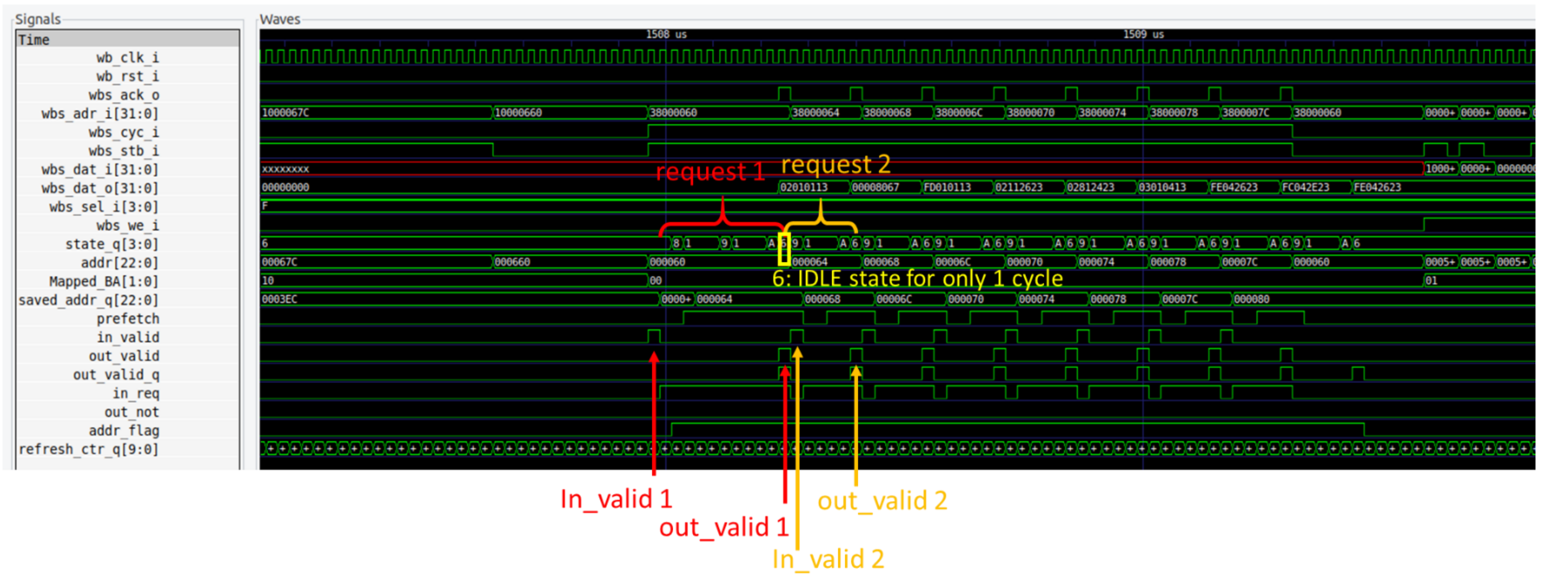
Output valid will be blocked by out_not

**addr_flag**: goes 1 means there's an address sent to SDRAM and waiting for the output data.

```
addr_flag_n = (in_req & (state_q == IDLE) & !ready_q) ? 1'b1 : out_valid_q ? 1'b0 : addr_flag;
```

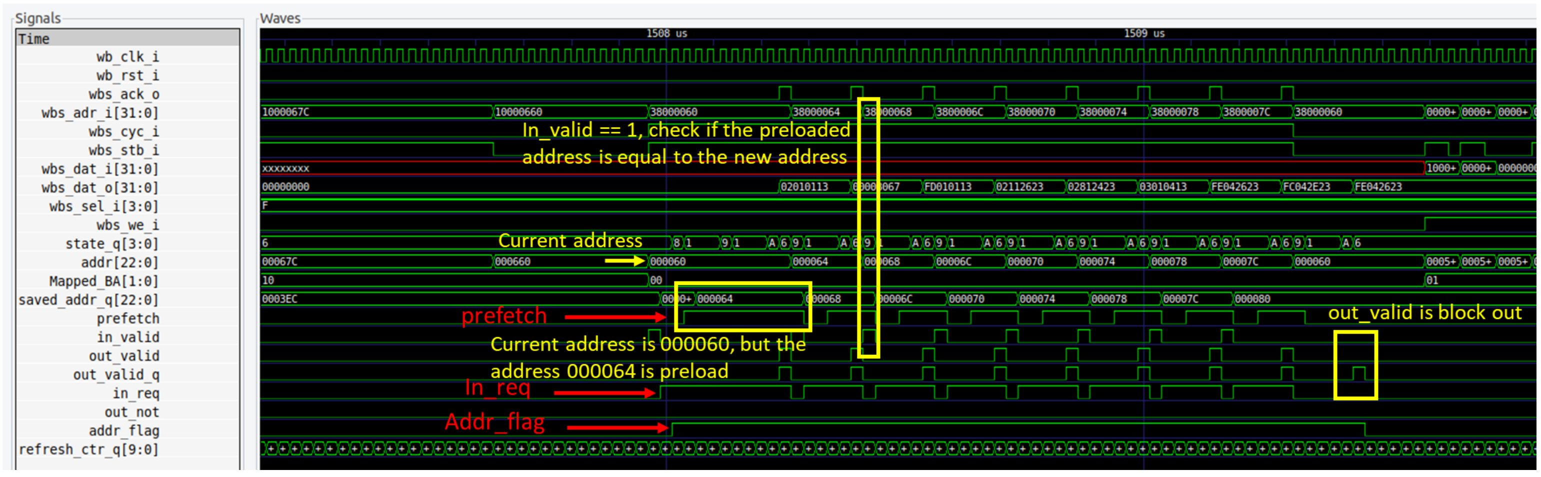
The condition that address was send to SDRAM controller

Two cycles are saved after adding the prefetch scheme.



6: IDLE state for only 1 cycle

The detail operation of the prefetch scheme.

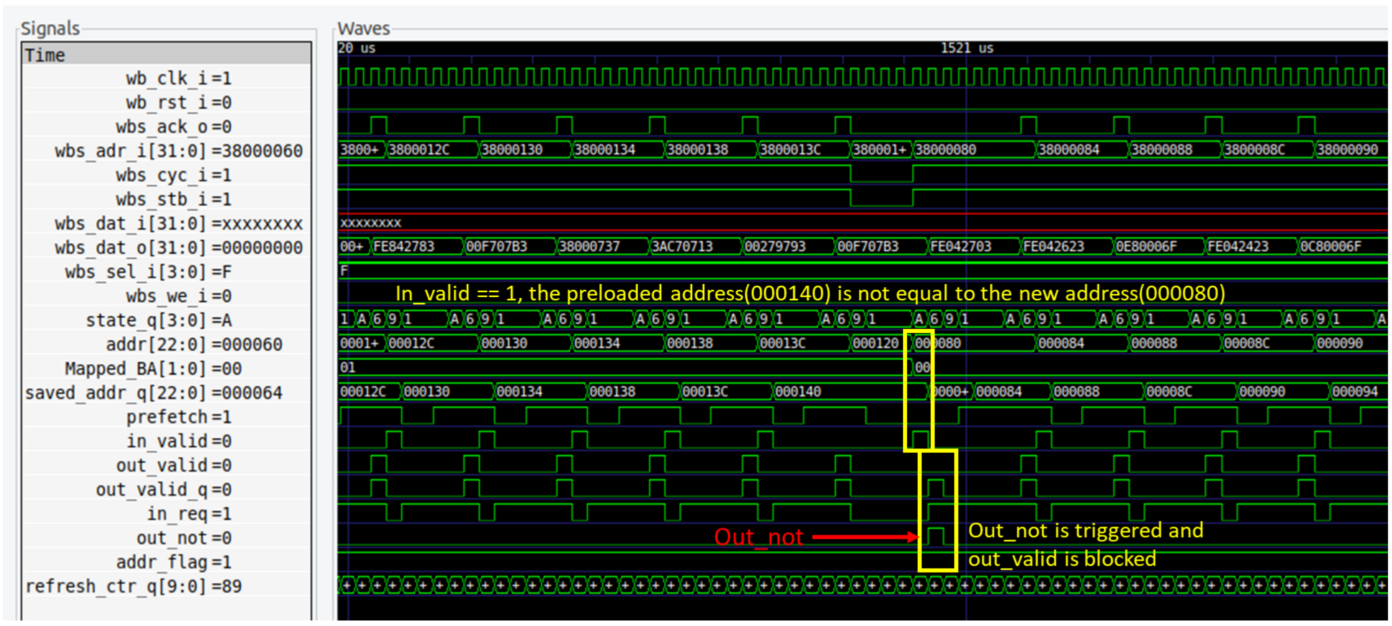


In_valid == 1, check if the preloaded address is equal to the new address

Current address is 000060, but the address 000064 is preload

out_valid is block out

Example of the miss situation.



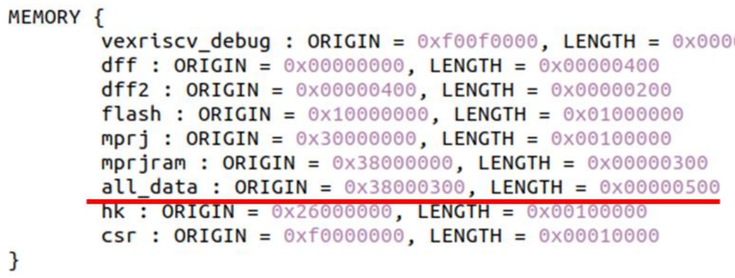
wb_clk_i =1
wb_rst_i =0
wbs_ack_o =0
wbs_adr_i[31:0] =38000060
wbs_cyc_i =1
wbs_stb_i =1
wbs_dat_i[31:0] =xxxxxxxx
wbs_dat_o[31:0] =00000000
wbs_sel_i[3:0] =F
wbs_we_i =0
state_q[3:0] =A
addr[22:0] =000060
Mapped_BA[1:0] =00
saved_addr_q[22:0] =000064
prefetch =1
in_valid =0
out_valid =0
out_valid_q =0
in_req =1
out_not =0
addr_flag =1
refresh_ctr_q[9:0] =89

In_valid == 1, the preloaded address(000140) is not equal to the new address(000080)

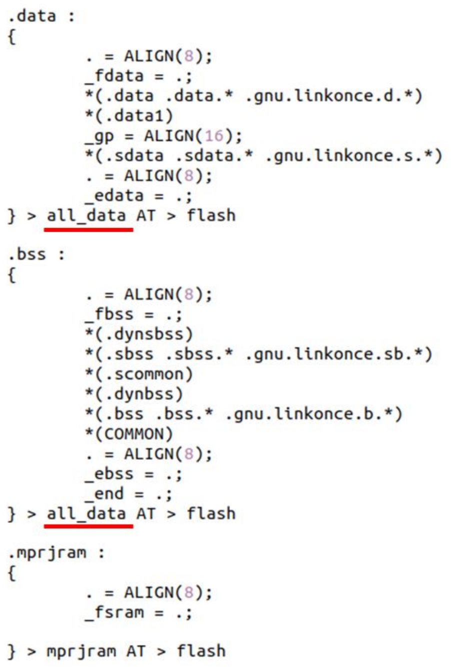
Out_not is triggered and out_valid is blocked

## Modify the linker to load address/data in two different banks

In sections.lds, we modify the address mapping such that the data will be located in 0x38000300 ~ 0x38000800 and the firmware code will be located in 0x38000000 ~ 0x38000300. Thus, the data and the code will automatically be located into bank 3 (data) and bank 0~2 (code).



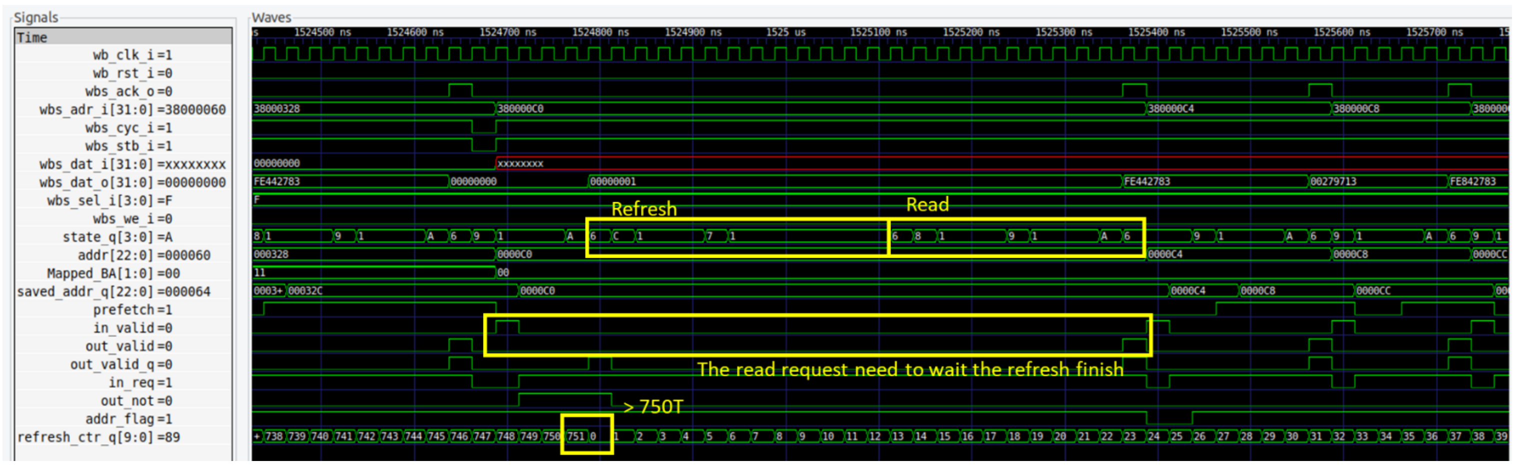
```
MEMORY {
        vexriscv_debug : ORIGIN = 0xf00f0000, LENGTH = 0x00000100
        dff : ORIGIN = 0x00000000, LENGTH = 0x00000400
        dff2 : ORIGIN = 0x00000400, LENGTH = 0x00000200
        flash : ORIGIN = 0x10000000, LENGTH = 0x01000000
        mprj : ORIGIN = 0x30000000, LENGTH = 0x00100000
        mprjram : ORIGIN = 0x38000000, LENGTH = 0x00000300
        all_data : ORIGIN = 0x38000300, LENGTH = 0x00000500
        hk : ORIGIN = 0x26000000, LENGTH = 0x00100000
        csr : ORIGIN = 0xf0000000, LENGTH = 0x00010000
}
```



```
 .data :
 {
        . = ALIGN(8);
        _fdata = .;
        *(.data .data.* .gnu.linkonce.d.*)
        *(.data1)
        _gp = ALIGN(16);
        *(.sdata .sdata.* .gnu.linkonce.s.*)
        . = ALIGN(8);
        _edata = .;
 } > all_data AT > flash

 .bss :
 {
        . = ALIGN(8);
        _fbss = .;
        *(.dynsbss)
        *(.sbss .sbss.* .gnu.linkonce.sb.*)
        *(.scommon)
        *(.dynbss)
        *(.bss .bss.* .gnu.linkonce.b.*)
        *(COMMON)
        . = ALIGN(8);
        _ebss = .;
        _end = .;
 } > all_data AT > flash

 .mprjram :
 {
        . = ALIGN(8);
        _fsram = .;

 } > mprjram AT > flash
```

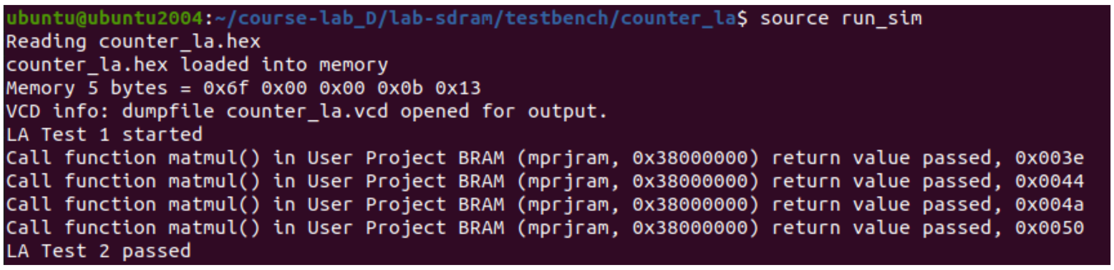
## Observe SDRAM access conflicts with SDRAM refresh (reduce the refresh period)

When the cycle > 750T:

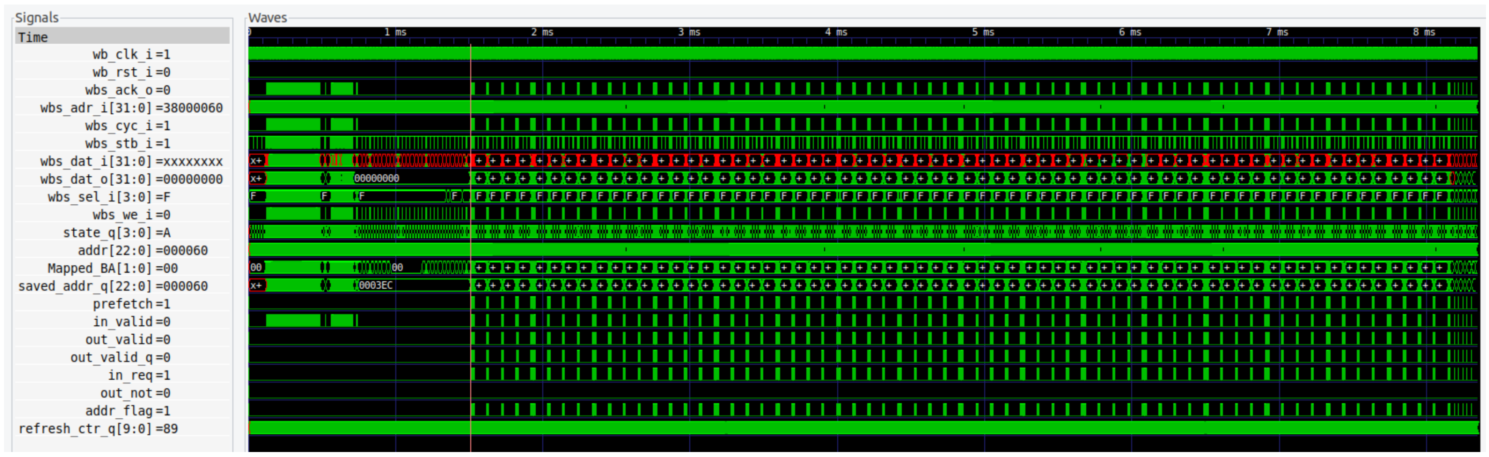


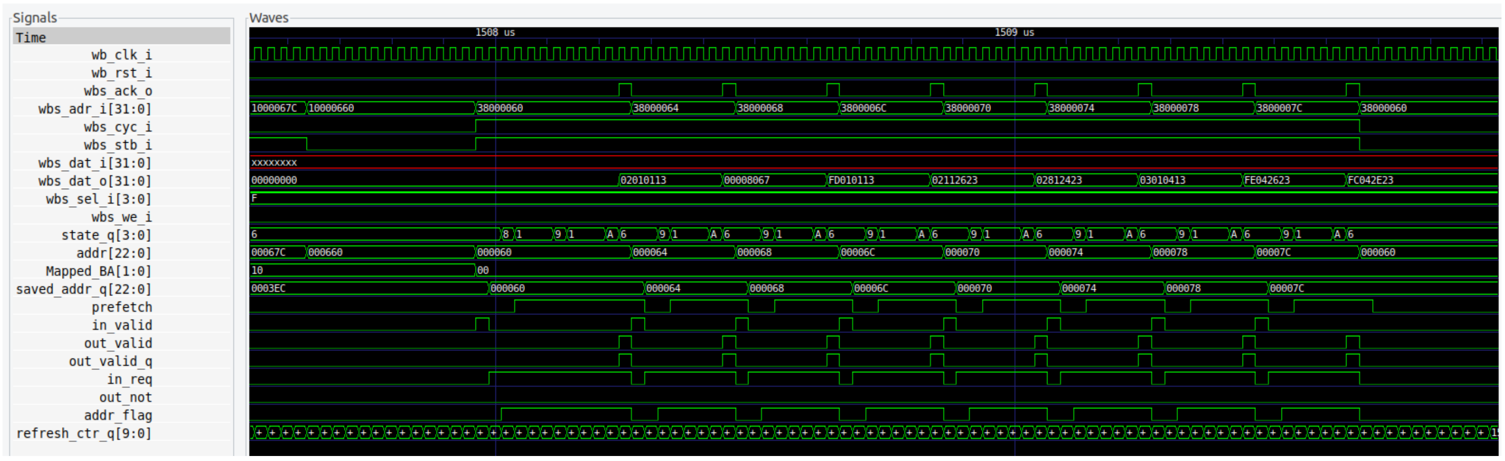
## Implementation results

Replace the adder with the matrix multiplier

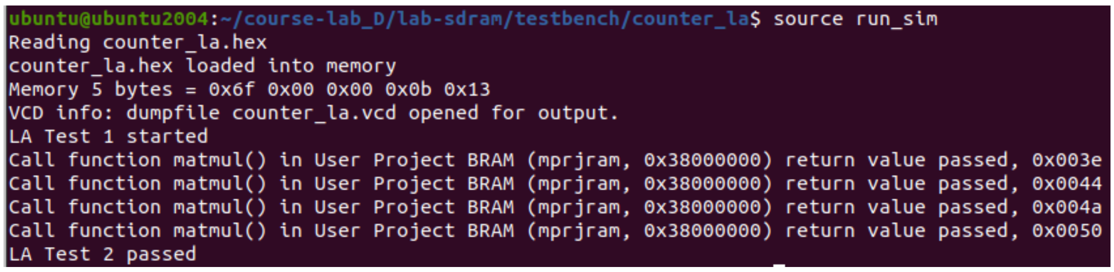


```
ubuntu@ubuntu2004:~/course-lab_D/lab-sdram/testbench/counter_la$ source run_sim
Reading counter_la.hex
counter_la.hex loaded into memory
Memory 5 bytes = 0x6f 0x00 0x00 0x0b 0x13
VCD info: dumpfile counter_la.vcd opened for output.
LA Test 1 started
Call function matmul() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x003e
Call function matmul() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x0044
Call function matmul() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x004a
Call function matmul() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x0050
LA Test 2 passed
```

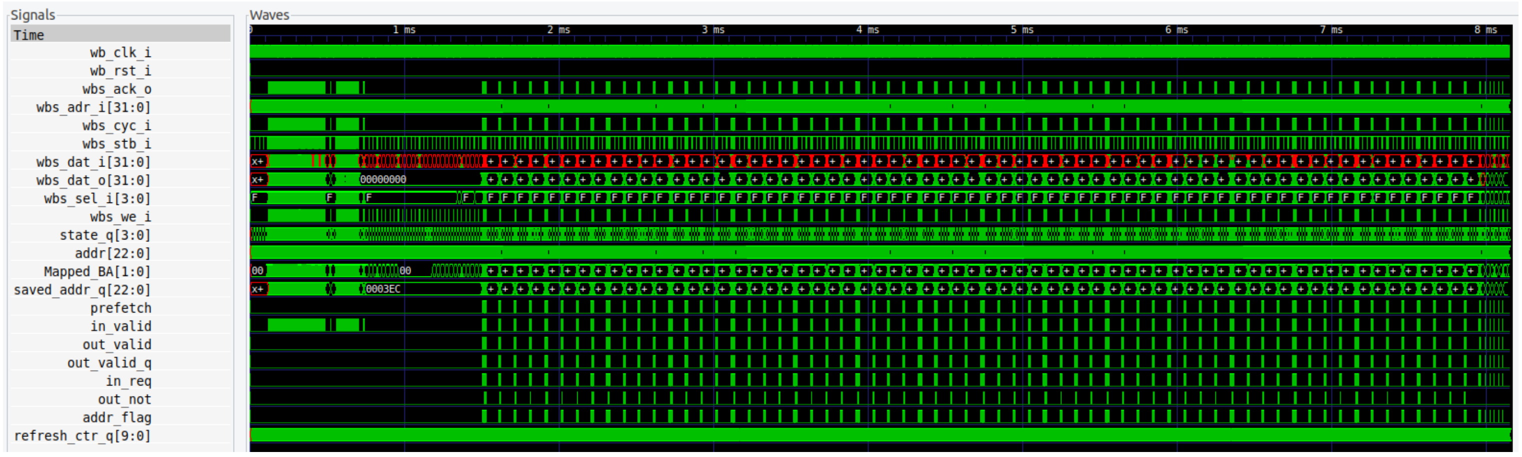


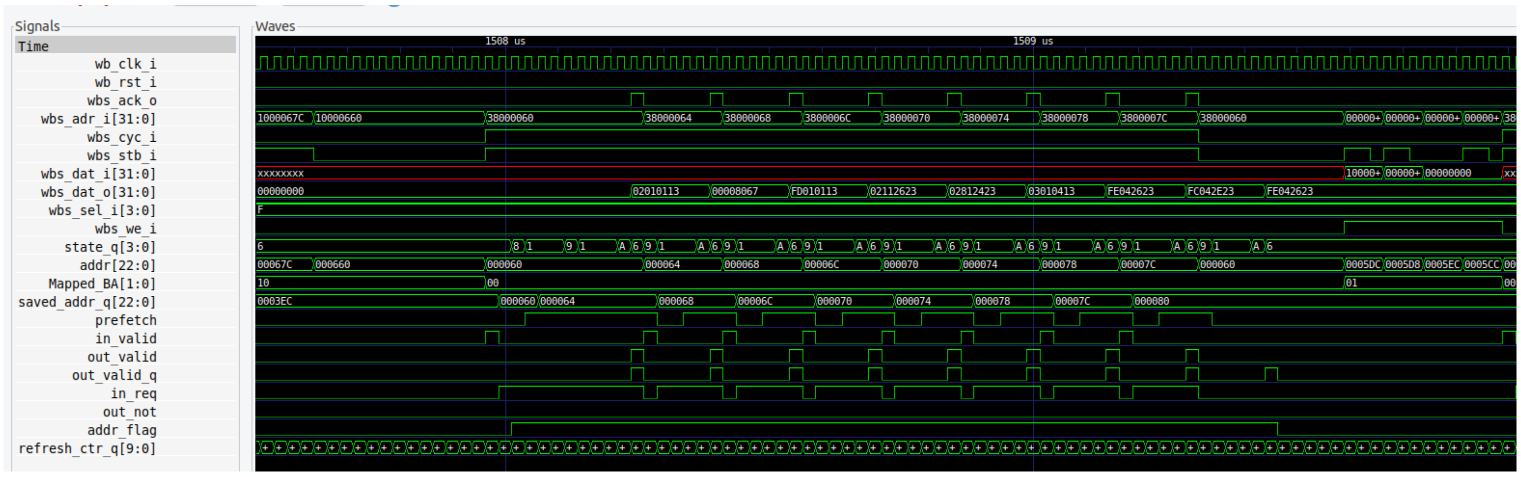


Add the prefetch scheme and bank interleave



```
ubuntu@ubuntu2004:~/course-lab_D/lab-sdram/testbench/counter_la$ source run_sim
Reading counter_la.hex
counter_la.hex loaded into memory
Memory 5 bytes = 0x6f 0x00 0x00 0x0b 0x13
VCD info: dumpfile counter_la.vcd opened for output.
LA Test 1 started
Call function matmul() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x003e
Call function matmul() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x0044
Call function matmul() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x004a
Call function matmul() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x0050
LA Test 2 passed
```


Signals
Waves
Time
1 ms
2 ms
3 ms
4 ms
5 ms
6 ms
7 ms
8 ms
wb_clk_i
wb_rst_i
wbs_ack_o
wbs_adr_i[31:0]
wbs_cyc_i
wbs_stb_i
wbs_dat_i[31:0]
wbs_dat_o[31:0]
wbs_sel_i[3:0]
wbs_we_i
state_q[3:0]
addr[22:0]
Mapped_BA[1:0]
saved_addr_q[22:0]
prefetch
in_valid
out_valid
out_valid_q
in_req
out_not
addr_flag
refresh_ctr_q[9:0]


Signals
Waves
Time
1508 us
1509 us
wb_clk_i
wb_rst_i
wbs_ack_o
wbs_adr_i[31:0]
wbs_cyc_i
wbs_stb_i
wbs_dat_i[31:0]
wbs_dat_o[31:0]
wbs_sel_i[3:0]
wbs_we_i
state_q[3:0]
addr[22:0]
Mapped_BA[1:0]
saved_addr_q[22:0]
prefetch
in_valid
out_valid
out_valid_q
in_req
out_not
addr_flag
refresh_ctr_q[9:0]
1000067C
10000660
38000060
38000064
38000068
3800006C
38000070
38000074
38000078
3800007C
38000060
02010113
00008067
FD010113
02112623
02812423
03010413
FE042623
FC042E23
FE042623