LabD

https://github.com/Josephood7/sdram

▼ Team member

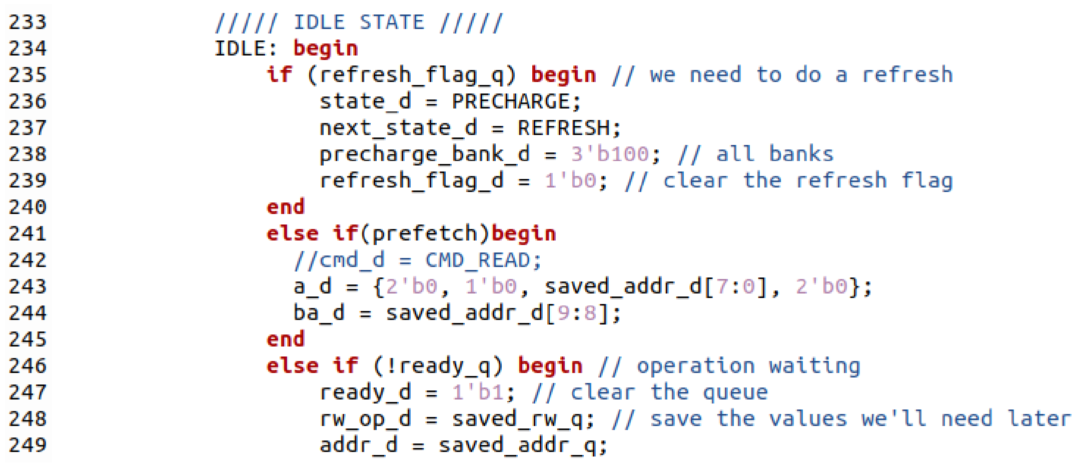
羅崇榮 R10522526

陳靖雯 R12922a10

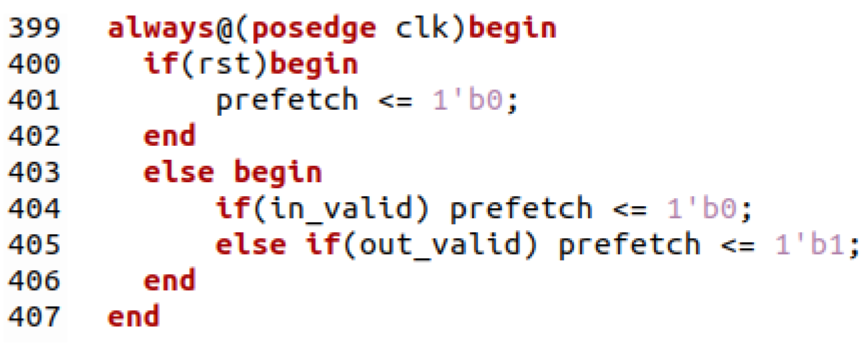
張景華 B09505021

▼ the SDRAM controller design, SDRAM bus protocol

> The controller interacts with the SDRAM bus protocol to realize the prefetch technique.



```
        ///// IDLE STATE /////
        IDLE: begin
            if (refresh_flag_q) begin // we need to do a refresh
                state_d = PRECHARGE;
                next_state_d = REFRESH;
                precharge_bank_d = 3'b100; // all banks
                refresh_flag_d = 1'b0; // clear the refresh flag
            end
            else if(prefetch)begin
              //cmd_d = CMD_READ;
              a_d = {2'b0, 1'b0, saved_addr_d[7:0], 2'b0};
              ba_d = saved_addr_d[9:8];
            end
            else if (!ready_q) begin // operation waiting
                ready_d = 1'b1; // clear the queue
                rw_op_d = saved_rw_q; // save the values we'll need later
                addr_d = saved_addr_q;
```



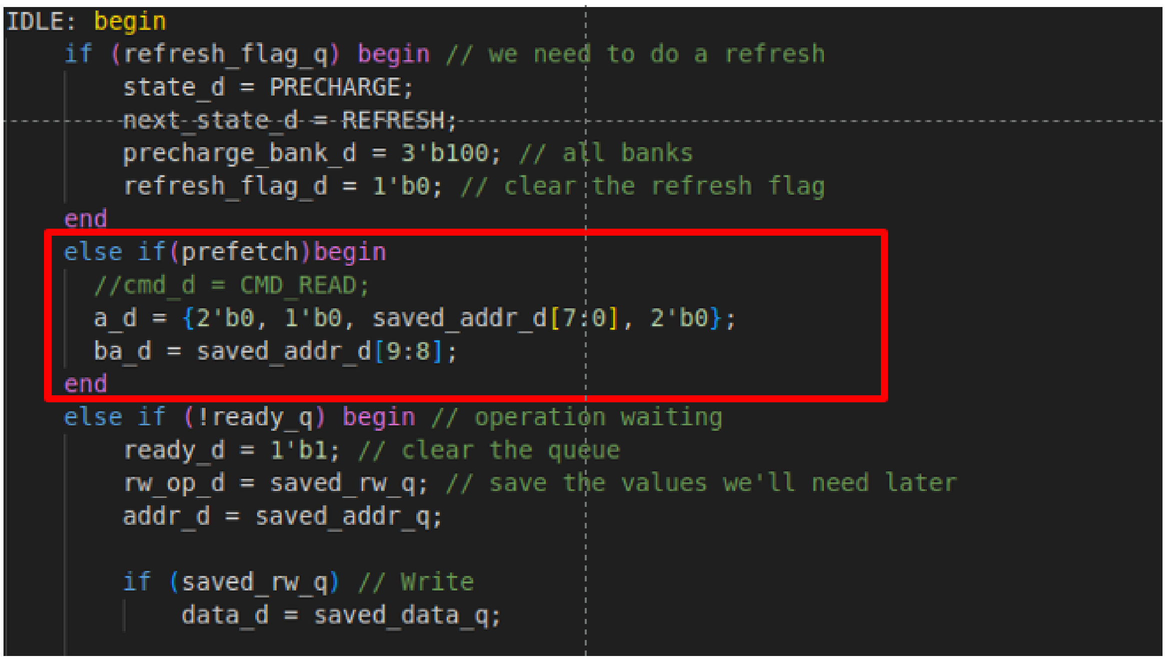
```
always@(posedge clk)begin
  if(rst)begin
      prefetch <= 1'b0;
  end
  else begin
      if(in_valid) prefetch <= 1'b0;
      else if(out_valid) prefetch <= 1'b1;
  end
end
```

▼ Introduce the prefetching scheme

> In the original framework, when the system is in the IDLE state, it needs to wait for three cycles before proceeding to the next step. However, in reality, once we output a piece of data, we can already start processing the next set of data. Therefore, we have introduced a prefetch flag. When out_valid is 1, the system will prefetch the address of the next data, so that in the IDLE state, it only needs to wait for two cycles, reducing the overall required cycles for the Read operation from the original 9 cycles to 8 cycles.

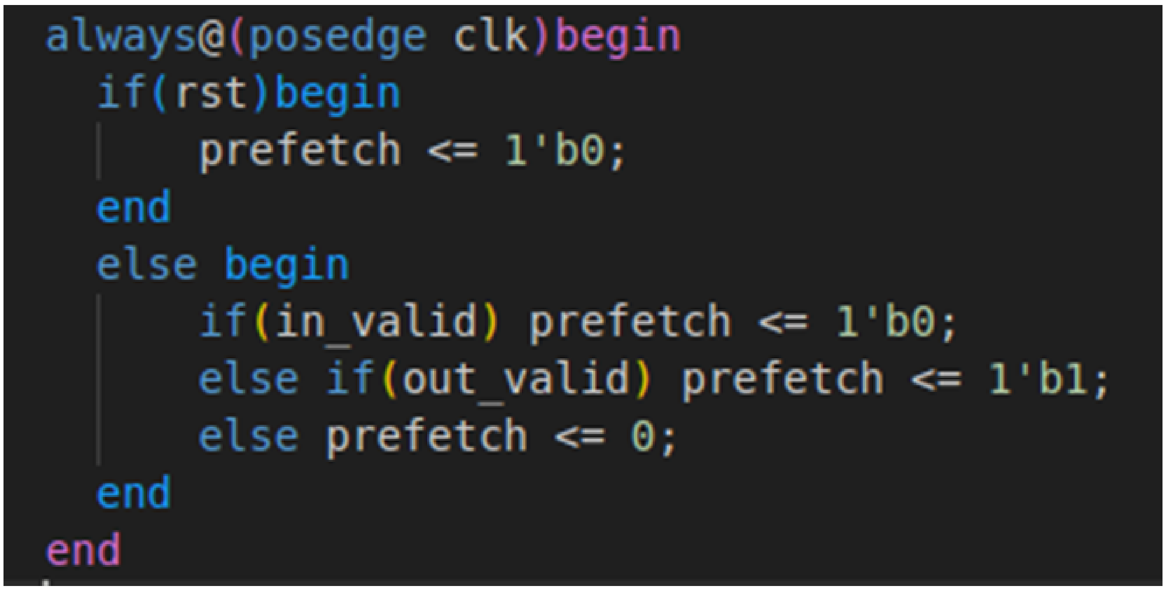
> When prefetching, put the read address and Bank address into SDRAM first. So that while it is accessed, we simply access it from SDRAM, which is near to the hardware.

- If the prefetch flag is raised, at IDLE state, send read address to the sdram

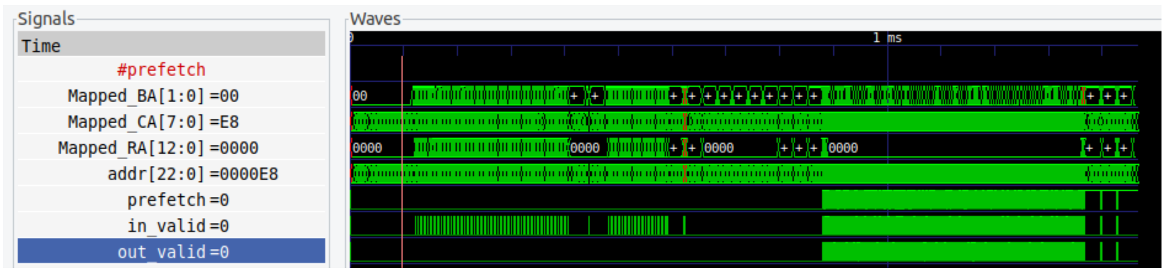


```
IDLE: begin
    if (refresh_flag_q) begin // we need to do a refresh
        state_d = PRECHARGE;
        next_state_d = REFRESH;
        precharge_bank_d = 3'b100; // all banks
        refresh_flag_d = 1'b0; // clear the refresh flag
    end
    else if(prefetch)begin
      //cmd_d = CMD_READ;
      a_d = {2'b0, 1'b0, saved_addr_d[7:0], 2'b0};
      ba_d = saved_addr_d[9:8];
    end
    else if (!ready_q) begin // operation waiting
        ready_d = 1'b1; // clear the queue
        rw_op_d = saved_rw_q; // save the values we'll need later
        addr_d = saved_addr_q;

        if (saved_rw_q) // Write
            data_d = saved_data_q;
```

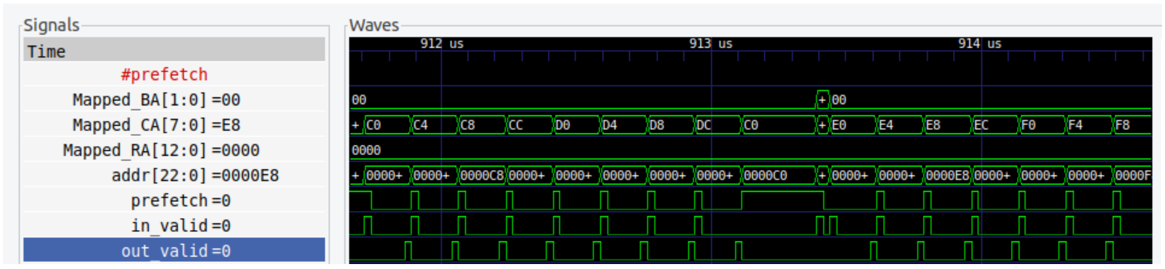
- Once *out_valid* is raised, start prefetch. When *in_valid*, stop prefetching.
- Add a prefetch flag



```verilog
always@(posedge clk)begin
   if(rst)begin
      prefetch <= 1'b0;
   end
   else begin
      if(in_valid) prefetch <= 1'b0;
      else if(out_valid) prefetch <= 1'b1;
      else prefetch <= 0;
   end
end
```



```
Signals
Time
#prefetch
Mapped_BA[1:0] =00
Mapped_CA[7:0] =E8
Mapped_RA[12:0] =0000
addr[22:0] =0000E8
prefetch =0
in_valid =0
out_valid =0
```

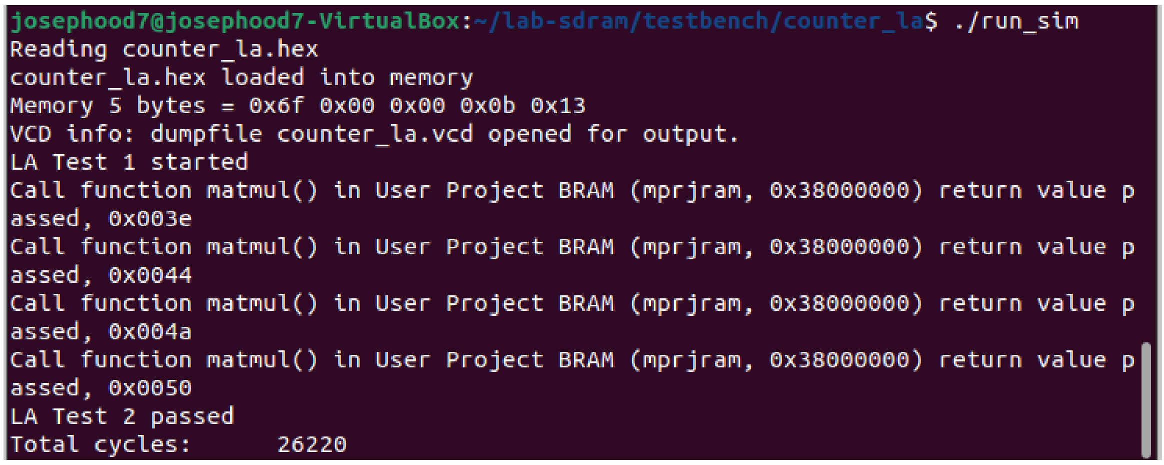


```
Signals
Time
#prefetch
Mapped_BA[1:0] =00
Mapped_CA[7:0] =E8
Mapped_RA[12:0] =0000
addr[22:0] =0000E8
prefetch =0
in_valid =0
out_valid =0
```

Waves
912 us
913 us
914 us
00
+ 00
+ C0 C4 C8 CC D0 D4 D8 DC C0 + E0 E4 E8 EC F0 F4 F8
0000
+ 0000+ 0000+ 0000C8 0000+ 0000+ 0000+ 0000+ 0000+ 0000C0 + 0000+ 0000+ 0000E8 0000+ 0000+ 0000+ 0000F

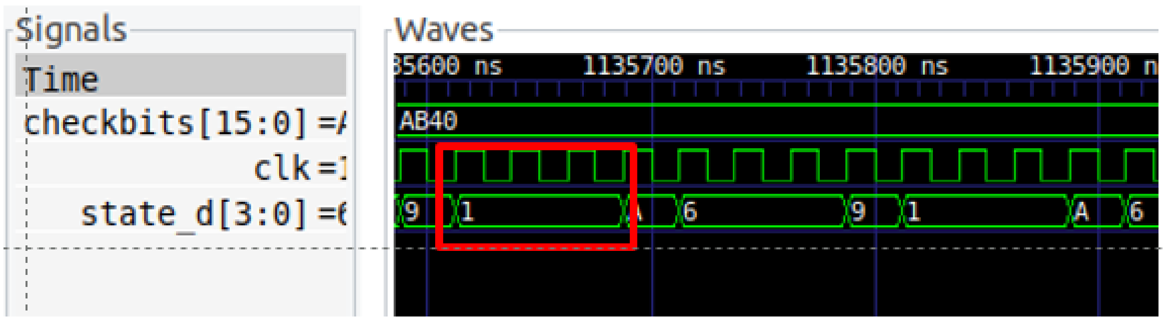
▼ prefetch result

- Our optimized matrix multiplication with -O1 compiler

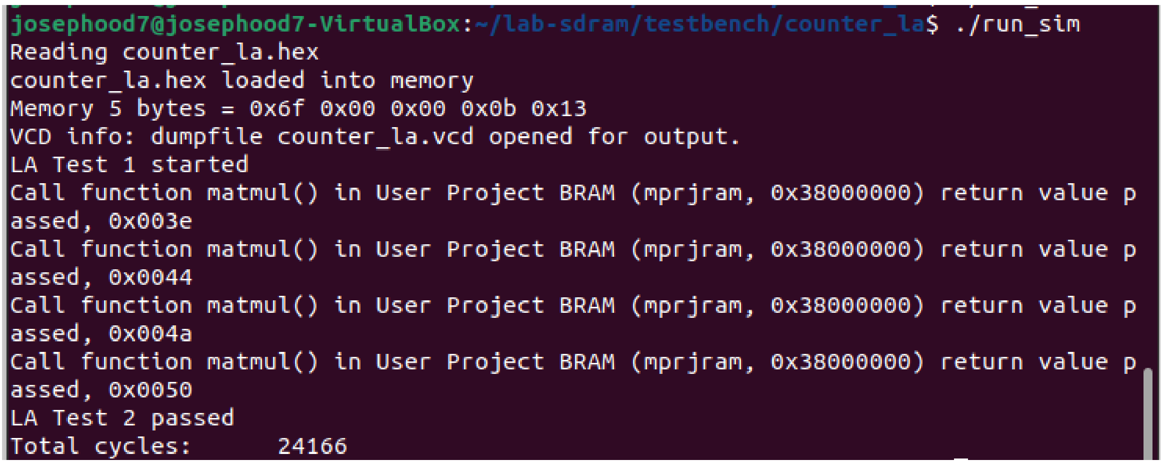


```
josephood7@josephood7-VirtualBox:~/lab-sdram/testbench/counter_la$ ./run_sim
Reading counter_la.hex
counter_la.hex loaded into memory
Memory 5 bytes = 0x6f 0x00 0x00 0x0b 0x13
VCD info: dumpfile counter_la.vcd opened for output.
LA Test 1 started
Call function matmul() in User Project BRAM (mprjram, 0x38000000) return value p
assed, 0x003e
Call function matmul() in User Project BRAM (mprjram, 0x38000000) return value p
assed, 0x0044
Call function matmul() in User Project BRAM (mprjram, 0x38000000) return value p
assed, 0x004a
Call function matmul() in User Project BRAM (mprjram, 0x38000000) return value p
assed, 0x0050
LA Test 2 passed
Total cycles:       26220
```

IDLE state spend 3 cycles

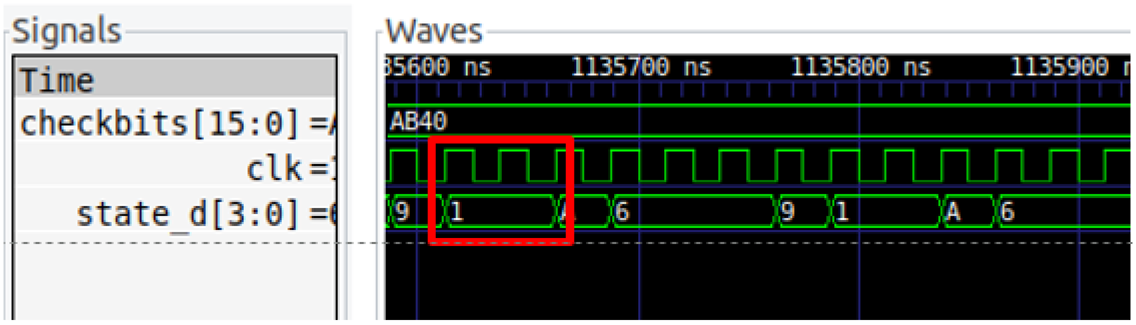


- After prefetch is introduced.
- Improved by 2k cycle.



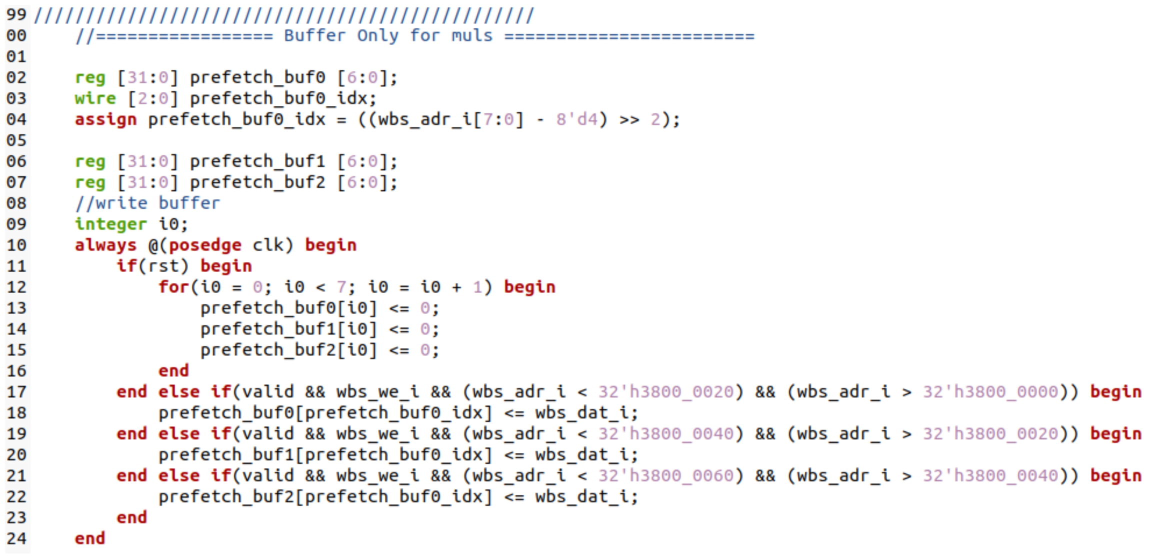
```
josephood7@josephood7-VirtualBox:~/lab-sdram/testbench/counter_la$ ./run_sim
Reading counter_la.hex
counter_la.hex loaded into memory
Memory 5 bytes = 0x6f 0x00 0x00 0x0b 0x13
VCD info: dumpfile counter_la.vcd opened for output.
LA Test 1 started
Call function matmul() in User Project BRAM (mprjram, 0x38000000) return value p
assed, 0x003e
Call function matmul() in User Project BRAM (mprjram, 0x38000000) return value p
assed, 0x0044
Call function matmul() in User Project BRAM (mprjram, 0x38000000) return value p
assed, 0x004a
Call function matmul() in User Project BRAM (mprjram, 0x38000000) return value p
assed, 0x0050
LA Test 2 passed
Total cycles:       24166
```

IDLE state spend 2 cycles

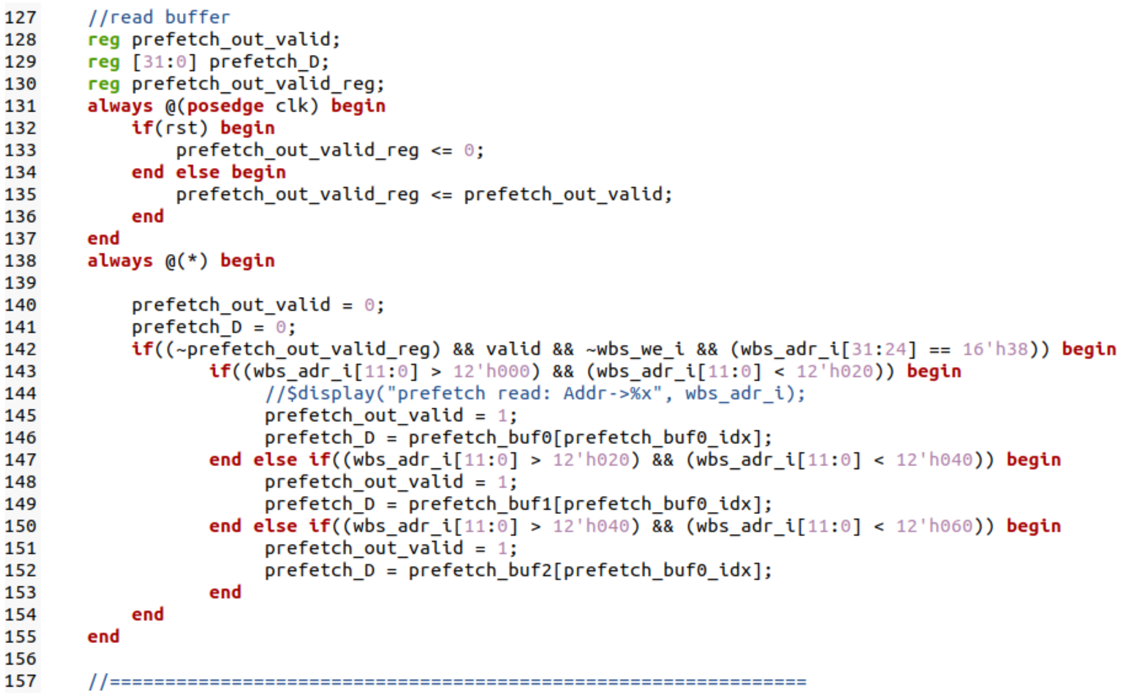


▼ prefetch - faster approach

> Instead of using *sdram*, we implement prefetch buffer in *counter_la*. The result is significantly faster than our previous approach.

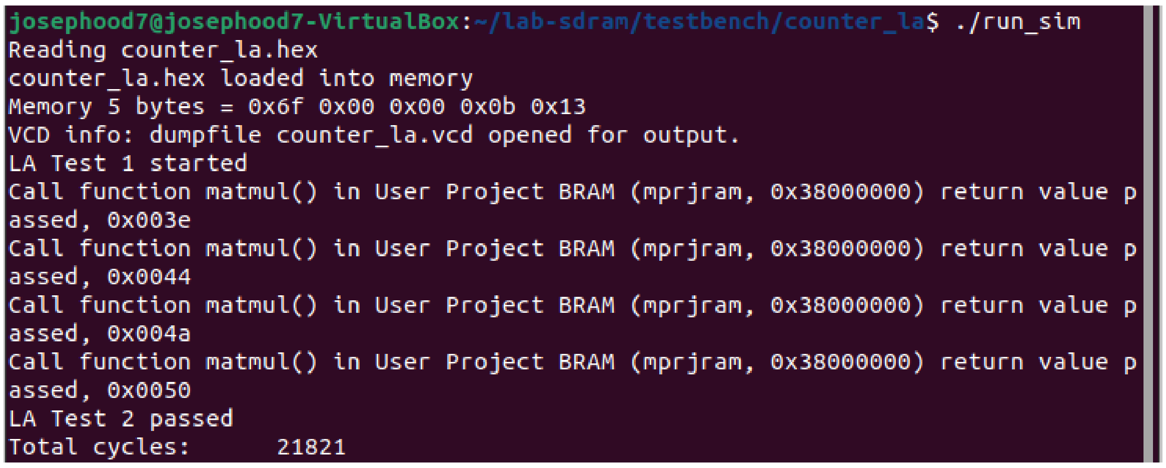


```
////////////////////////////////////////////////
    //================= Buffer Only for muls ========================

    reg [31:0] prefetch_buf0 [6:0];
    wire [2:0] prefetch_buf0_idx;
    assign prefetch_buf0_idx = ((wbs_adr_i[7:0] - 8'd4) >> 2);

    reg [31:0] prefetch_buf1 [6:0];
    reg [31:0] prefetch_buf2 [6:0];
    //write buffer
    integer i0;
    always @(posedge clk) begin
        if(rst) begin
            for(i0 = 0; i0 < 7; i0 = i0 + 1) begin
                prefetch_buf0[i0] <= 0;
                prefetch_buf1[i0] <= 0;
                prefetch_buf2[i0] <= 0;
            end
        end else if(valid && wbs_we_i && (wbs_adr_i < 32'h3800_0020) && (wbs_adr_i > 32'h3800_0000)) begin
            prefetch_buf0[prefetch_buf0_idx] <= wbs_dat_i;
        end else if(valid && wbs_we_i && (wbs_adr_i < 32'h3800_0040) && (wbs_adr_i > 32'h3800_0020)) begin
            prefetch_buf1[prefetch_buf0_idx] <= wbs_dat_i;
        end else if(valid && wbs_we_i && (wbs_adr_i < 32'h3800_0060) && (wbs_adr_i > 32'h3800_0040)) begin
            prefetch_buf2[prefetch_buf0_idx] <= wbs_dat_i;
        end
    end
```

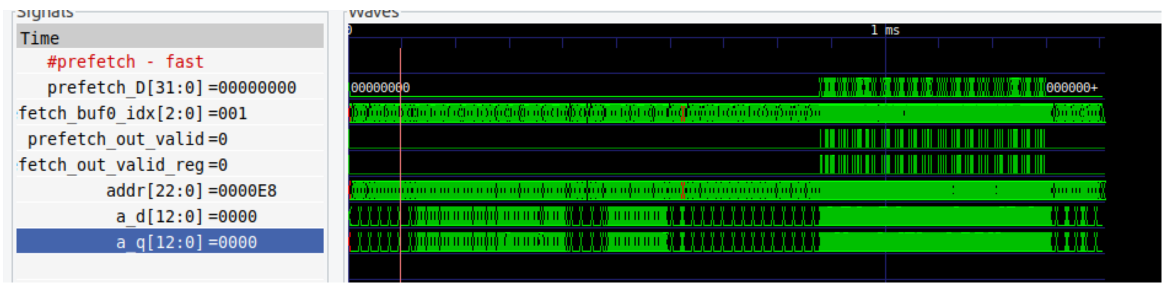


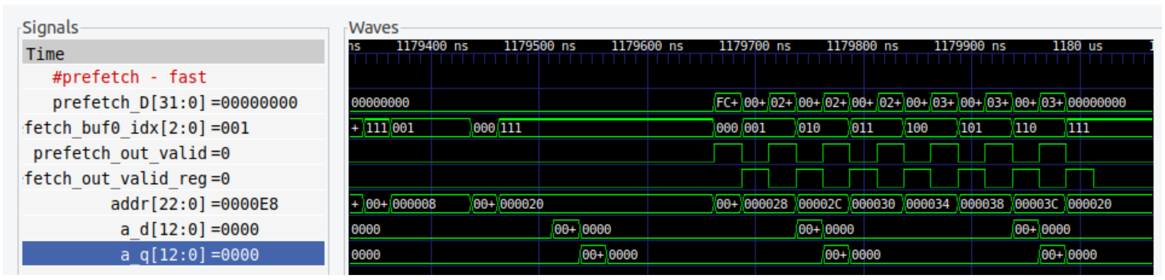
```
//read buffer
reg prefetch_out_valid;
reg [31:0] prefetch_D;
reg prefetch_out_valid_reg;
always @(posedge clk) begin
    if(rst) begin
        prefetch_out_valid_reg <= 0;
    end else begin
        prefetch_out_valid_reg <= prefetch_out_valid;
    end
end
always @(*) begin

    prefetch_out_valid = 0;
    prefetch_D = 0;
    if((~prefetch_out_valid_reg) && valid && ~wbs_we_i && (wbs_adr_i[31:24] == 16'h38)) begin
           if((wbs_adr_i[11:0] > 12'h000) && (wbs_adr_i[11:0] < 12'h020)) begin
                //$display("prefetch read: Addr->%x", wbs_adr_i);
                prefetch_out_valid = 1;
                prefetch_D = prefetch_buf0[prefetch_buf0_idx];
           end else if((wbs_adr_i[11:0] > 12'h020) && (wbs_adr_i[11:0] < 12'h040)) begin
                prefetch_out_valid = 1;
                prefetch_D = prefetch_buf1[prefetch_buf0_idx];
           end else if((wbs_adr_i[11:0] > 12'h040) && (wbs_adr_i[11:0] < 12'h060)) begin
                prefetch_out_valid = 1;
                prefetch_D = prefetch_buf2[prefetch_buf0_idx];
           end
    end
end

//================================================================
```

- 2 times faster than the previous prefetch approach.



```
josephood7@josephood7-VirtualBox:~/lab-sdram/testbench/counter_la$ ./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 p
assed, 0x003e
Call function matmul() in User Project BRAM (mprjram, 0x38000000) return value p
assed, 0x0044
Call function matmul() in User Project BRAM (mprjram, 0x38000000) return value p
assed, 0x004a
Call function matmul() in User Project BRAM (mprjram, 0x38000000) return value p
assed, 0x0050
LA Test 2 passed
Total cycles:      21821
```



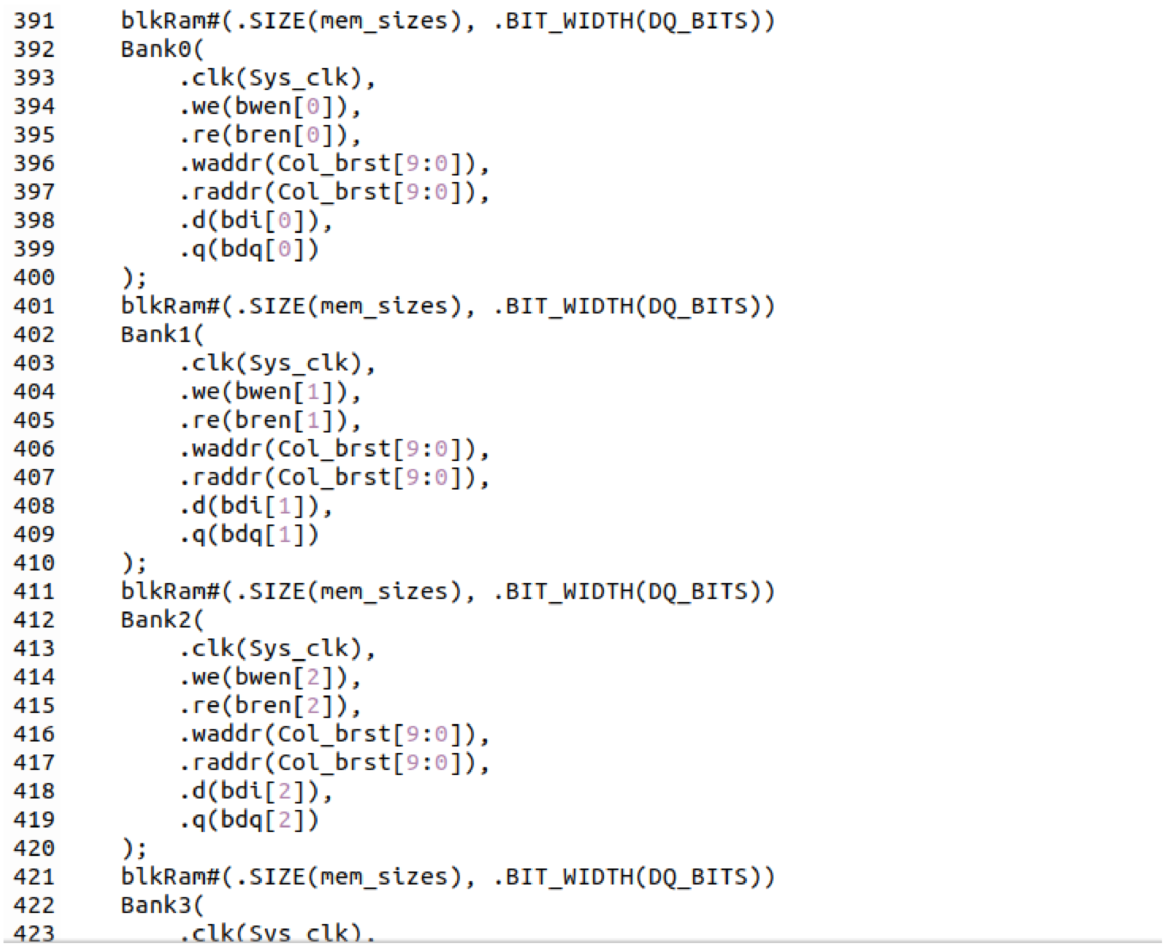


```
Signals
Time
   #prefetch - fast
   prefetch_D[31:0] =00000000
fetch_buf0_idx[2:0] =001
 prefetch_out_valid =0
fetch_out_valid_reg =0
         addr[22:0] =0000E8
          a_d[12:0] =0000
          a_q[12:0] =0000
```

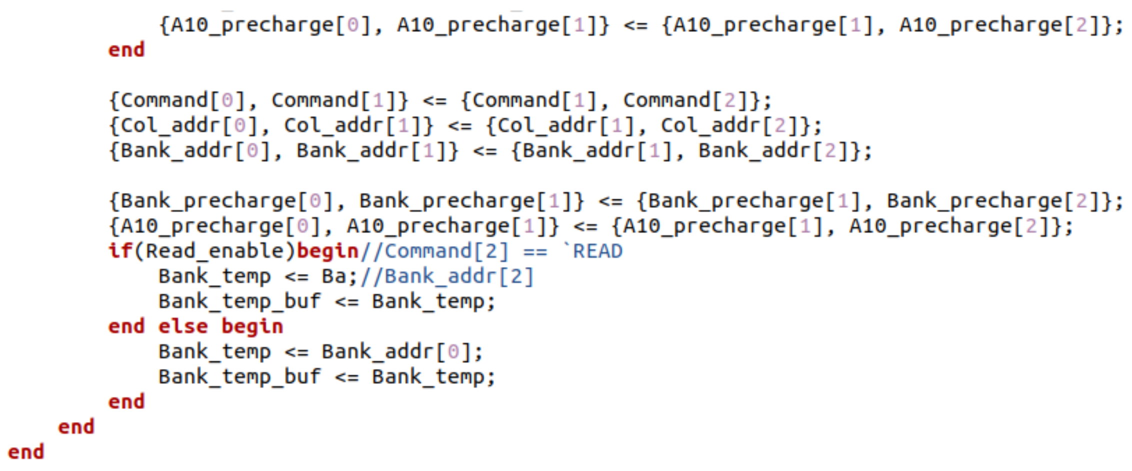
▼ Introduce the bank interleave for code and data

> Basically, according to the address[9:8], which is the Bank Address. We can determine which bank does the data or code stores.

```
always@(*)begin
    bwen = 4'b0000;
    bren = 4'b0000;
    if(Data_in_enable)begin//Data_in_enable||Data_out_enable
        case(Bank_temp)
            2'b00:begin
                bwen = 4'b0001;
                bren = 4'b0000;
            end
            2'b01:begin
                bwen = 4'b0010;
                bren = 4'b0000;
            end
            2'b10:begin
                bwen = 4'b0100;
                bren = 4'b0000;
            end
            2'b11:begin
                bwen = 4'b1000;
                bren = 4'b0000;
            end
        endcase
    end else if(Command[2] == `READ)begin//Data_out_enable, Command[1] == `READ
        case(Bank_temp)
            2'b00:begin
                bwen = 4'b0000;
                bren = 4'b0001;
            end
            2'b01:begin
                bwen = 4'b0000;
                bren = 4'b0010;
            end
            2'b10:begin
                bwen = 4'b0000;
                bren = 4'b0100;
            end
            2'b11:begin
                bwen = 4'b0000;
                bren = 4'b1000;
            end
        endcase
    end
end
```



```
blkRam#(.SIZE(mem_sizes), .BIT_WIDTH(DQ_BITS))
Bank0(
    .clk(Sys_clk),
    .we(bwen[0]),
    .re(bren[0]),
    .waddr(Col_brst[9:0]),
    .raddr(Col_brst[9:0]),
    .d(bdi[0]),
    .q(bdq[0])
);
blkRam#(.SIZE(mem_sizes), .BIT_WIDTH(DQ_BITS))
Bank1(
    .clk(Sys_clk),
    .we(bwen[1]),
    .re(bren[1]),
    .waddr(Col_brst[9:0]),
    .raddr(Col_brst[9:0]),
    .d(bdi[1]),
    .q(bdq[1])
);
blkRam#(.SIZE(mem_sizes), .BIT_WIDTH(DQ_BITS))
Bank2(
    .clk(Sys_clk),
    .we(bwen[2]),
    .re(bren[2]),
    .waddr(Col_brst[9:0]),
    .raddr(Col_brst[9:0]),
    .d(bdi[2]),
    .q(bdq[2])
);
blkRam#(.SIZE(mem_sizes), .BIT_WIDTH(DQ_BITS))
Bank3(
    .clk(Sys_clk),
```

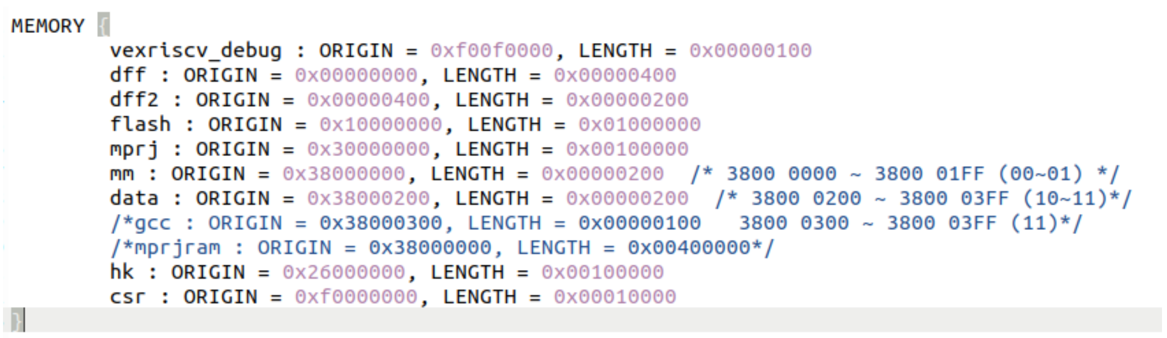


```
            {A10_precharge[0], A10_precharge[1]} <= {A10_precharge[1], A10_precharge[2]};
        end

        {Command[0], Command[1]} <= {Command[1], Command[2]};
        {Col_addr[0], Col_addr[1]} <= {Col_addr[1], Col_addr[2]};
        {Bank_addr[0], Bank_addr[1]} <= {Bank_addr[1], Bank_addr[2]};

        {Bank_precharge[0], Bank_precharge[1]} <= {Bank_precharge[1], Bank_precharge[2]};
        {A10_precharge[0], A10_precharge[1]} <= {A10_precharge[1], A10_precharge[2]};
        if(Read_enable)begin//Command[2] == `READ
            Bank_temp <= Ba;//Bank_addr[2]
            Bank_temp_buf <= Bank_temp;
        end else begin
            Bank_temp <= Bank_addr[0];
            Bank_temp_buf <= Bank_temp;
        end
    end
end
```

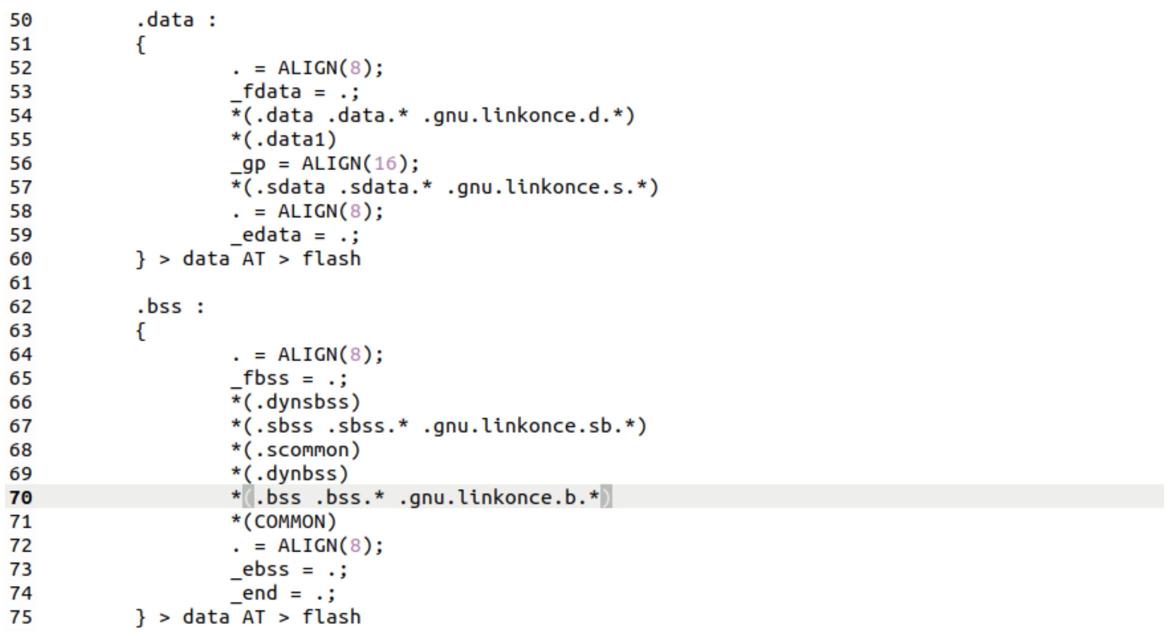
▼ Introduce how to modify the linker to load address/data in two different bank

> The bank address is specified as add[9:8], which is the Ba in 0x3800 0(Ba)00.

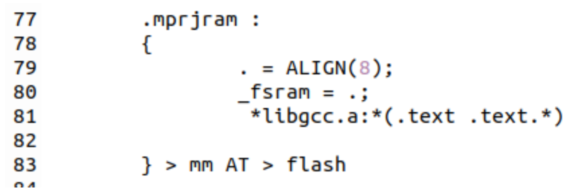
> Thus, we allocate 0000~01FF to the code and 0200~03FF to the data.



```
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
        mm : ORIGIN = 0x38000000, LENGTH = 0x00000200   /* 3800 0000 ~ 3800 01FF (00~01) */
        data : ORIGIN = 0x38000200, LENGTH = 0x00000200   /* 3800 0200 ~ 3800 03FF (10~11)*/
        /*gcc : ORIGIN = 0x38000300, LENGTH = 0x00000100   3800 0300 ~ 3800 03FF (11)*/
        /*mprjram : ORIGIN = 0x38000000, LENGTH = 0x00400000*/
        hk : ORIGIN = 0x26000000, LENGTH = 0x00100000
        csr : ORIGIN = 0xf0000000, LENGTH = 0x00010000
}
```



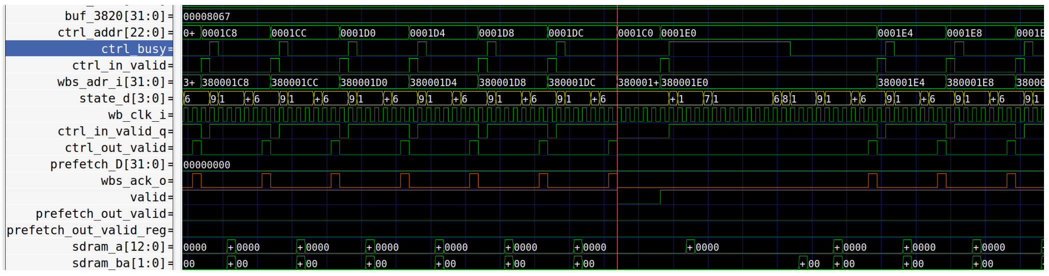
```
        .data :
        {
                . = ALIGN(8);
                _fdata = .;
                *(.data .data.* .gnu.linkonce.d.*)
                *(.data1)
                _gp = ALIGN(16);
                *(.sdata .sdata.* .gnu.linkonce.s.*)
                . = ALIGN(8);
                _edata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

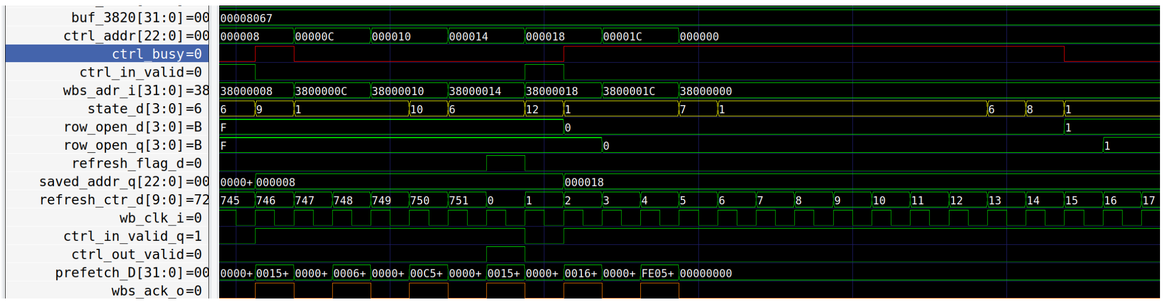


```
        .mprjram :
        {
                . = ALIGN(8);
                _fsram = .;
                 *libgcc.a:*(.text .text.*)

        } > mm AT > flash
```

▼ Observe SDRAM access conflicts with SDRAM refresh

- sdram refresh
  - if SDRAM doesn't conduct read or write operation, it would need to enter the refresh state, which is the state 7 in the following waveform.





In the above example, we could find that due to *refresh_ctrl_d* has reached 750, the state first enter the precharge state first, then it enters the refresh state.