Final Project:

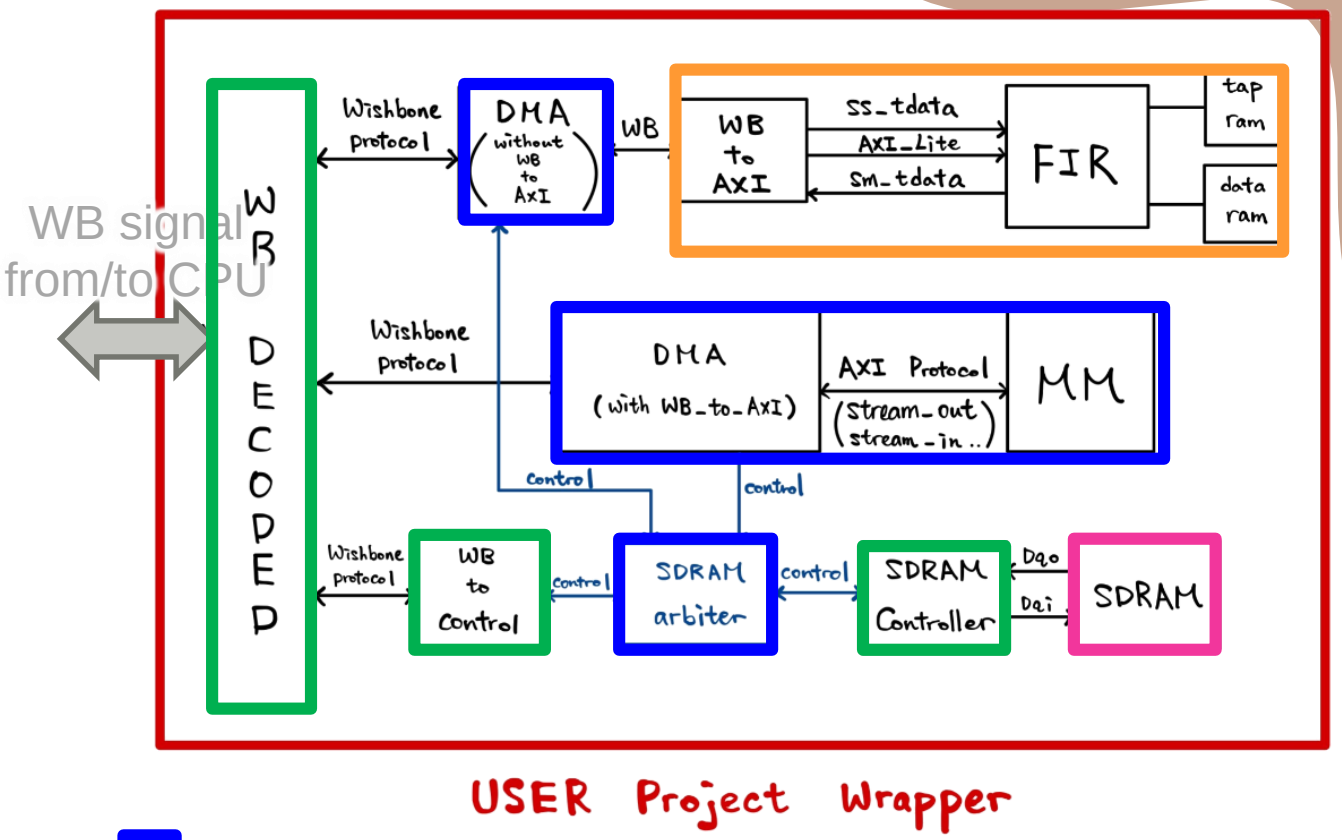
Q & A session

Group 12

112501538 葉承泓、 111063548 蕭方凱、111061624 尤弘瑋

Block Diagram

- Our final project is target at:
  1. Concurrently execution of all workloads & acceleration of MM
     → MM hardware accelerator
  2. Reduce FIR & MM latency
     → add DMA
  3. Reduce instruction count
     → compile with –Os flag
  4. Make use of bank interleaved between code and data → SDRAM
  5. Reduce SDRAM average latency
     → SDRAM with prefetch
  6. Multiple request to SDRAM
     → arranged by SDRAM arbiter



USER Project Wrapper

- Our new work
- Our previous work
- Modified from source code / our previous work
- Provided

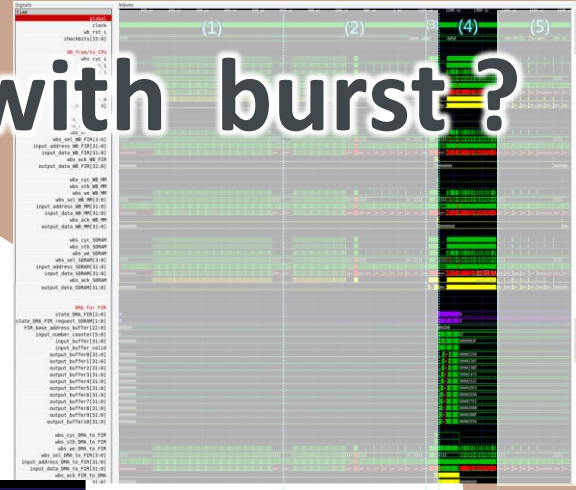
Q1: Explain why Qsort in CPU?

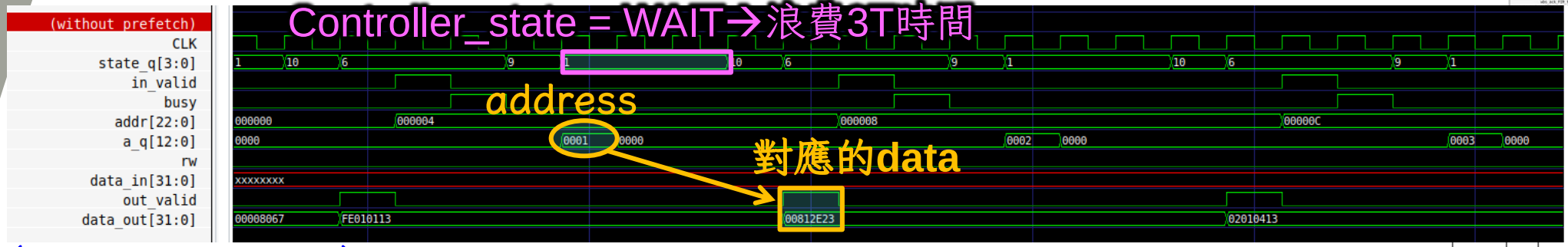
- FIR (with hardware & DMA & SDRAM)：
  - 總共 3058 個clock cycles
  - Compare to lab4-2：FIR with software：1510415個clock cycles
  - Compare to lab4-2：FIR with hardware：551675個clock cycles
- MM (hardware accelerator)
  - 總共 195 個clock cycles
  - ideally：48個clock cycles
  - Compare to lab6：MM with software：203852個clock cycles
- Qsort (firmware)
  - 總共 8305 個clock cycles
  - Compare to lab6：Qsort with software (without improvement techniques)：97082個clock cycles

→ including –Os、SDRAM (prefetch)

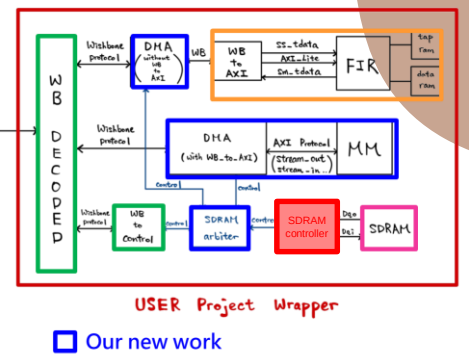
Q2: Explain prefetch mechanism, combine with burst?

- SDRAM controller
  - prefetch：利用controller IDLE的時間

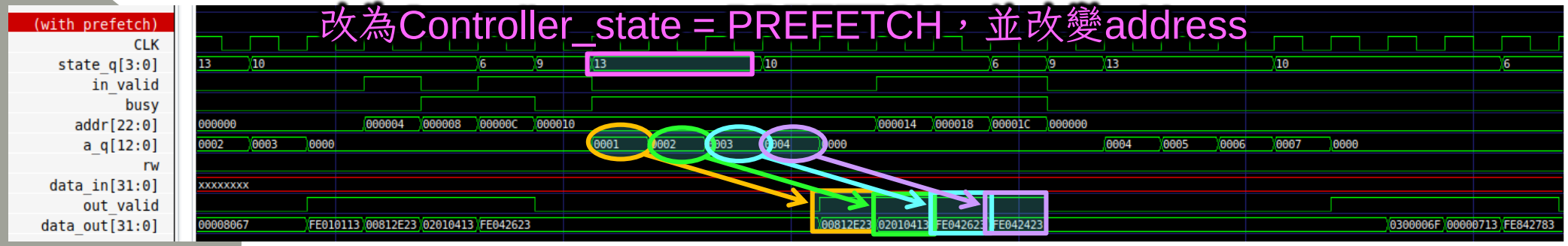




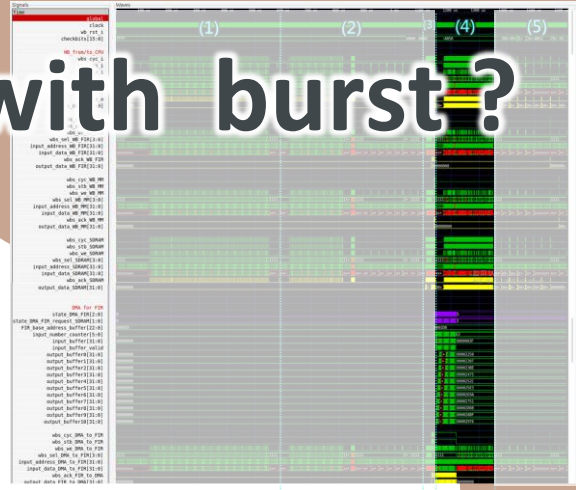
(without prefetch)



(with prefetch)

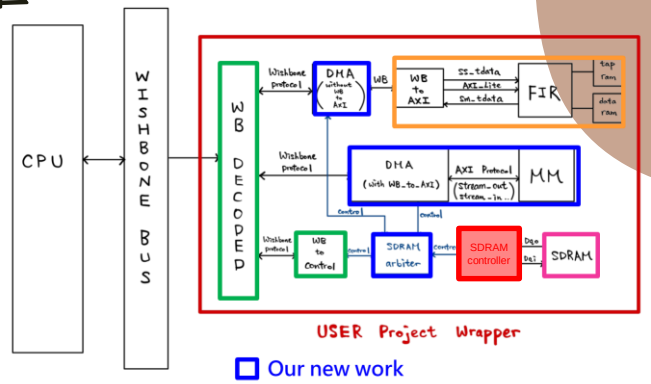


Q2: Explain prefetch mechanism, combine with burst?



- SDRAM controller
  - prefetch：
    1. 新增PREFETCH state，並replace上頁中的IDLE時段
    2. 將READ_RES state的時長加長3個cycle，分別輸出PREFETCH的3個 data → 一次request可輸出4筆data（with contiguous address）

→ 僅需多花費3個cycle（因PREFETCH state不會多花cycle），就可獲得額外3筆data →平均1個cycle就可以多獲得1筆data

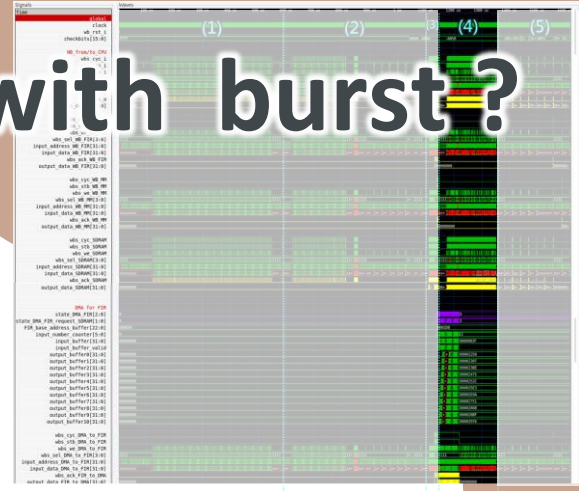



```
PREFETCH: begin

    cmd_d = CMD_READ;
    if (prefetch_step) begin
        a_d = a_q + 4; //16;
    end
    else begin
        a_d = a_q + 1; //4;
    end

    ba_d = ba_q;

    if (prefetch_counter == 2'd2) begin
        state_d = READ_RES;
        next_prefetch_counter = 0;
    end
    else begin
        state_d = PREFETCH;
        next_prefetch_counter = prefetch_counter + 1;
    end
end
```

```
READ_RES: begin
    data_d = dqi_q; // data_d by pass
    out_valid_d = 1'b1;
    /////state_d = IDLE;
    ////////////////////////// (Added by us) //////////////////////////
    next_prefetch_counter = prefetch_counter + 1;
    if (prefetch_counter == 2'd3) begin
        state_d = IDLE;
    end
    else begin
        state_d = READ_RES;
    end

    //////////////////////////////////////////////////////////////////
end
```

Q2: Explain prefetch mechanism, combine with burst?

- SDRAM controller
  - prefetch :



Request from MM

Request from FIR

Request address[0]

Request address[0]

Request from CPU

SDRAM Arbiter

(prefetch buffer)

CPU:

F

MM:

Prefetch buffer

AM request

2. Data0

→ return back to FIR/M

3. Data1

→ store into prefetch

4. Data2

→ store into prefetch

5. Data3

→ store into prefetch bu

Burst data[0:3]

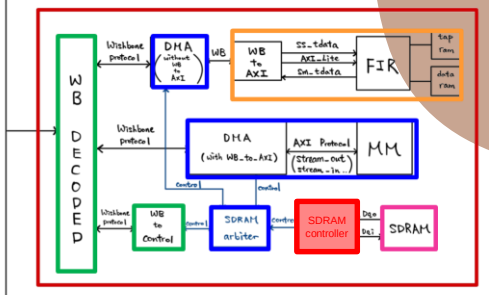
RAM troller

Request address[0:3]

SDRAM

CPU

WISHBONE BUS



USER Project Wrapper

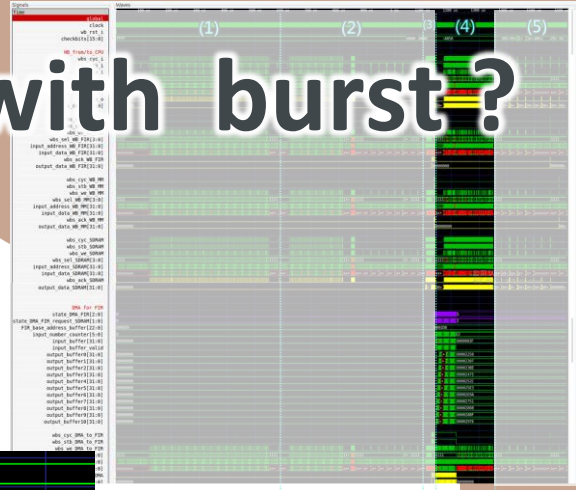
Our new work

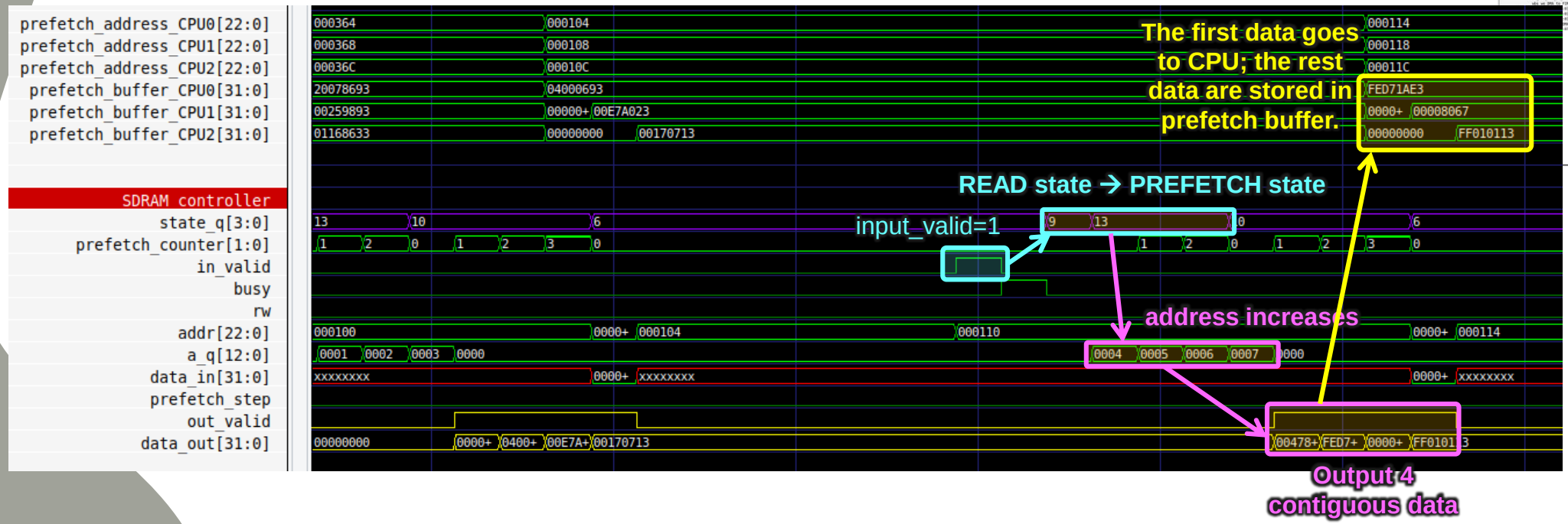
Prefetch

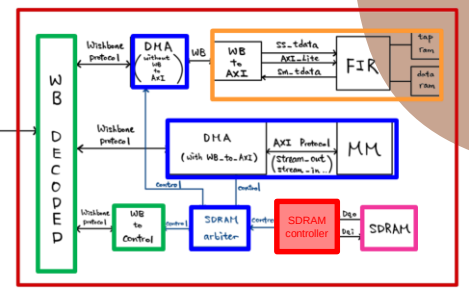
Burst

Q2: Explain prefetch mechanism, combine with burst?

- SDRAM controller
  - ➢ prefetch：下圖以CPU request為例



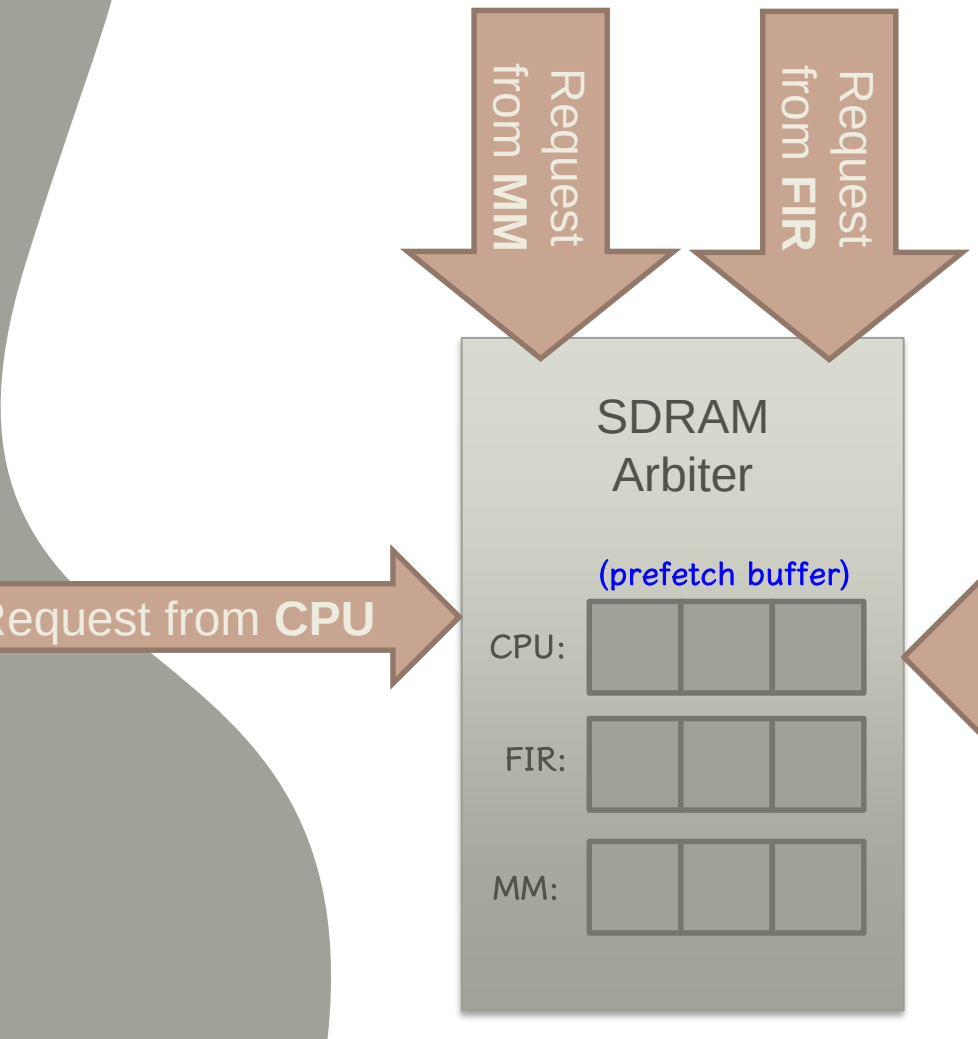


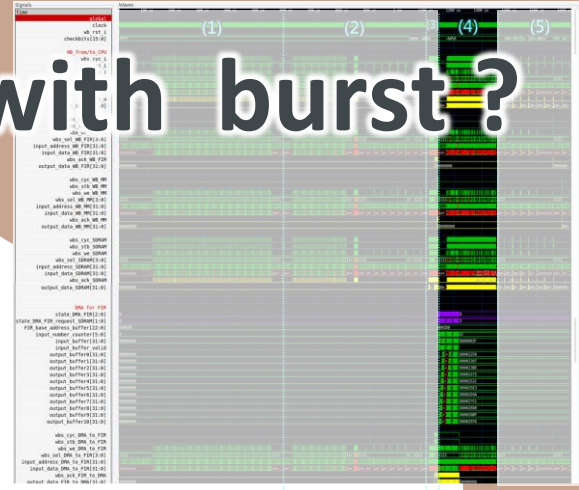


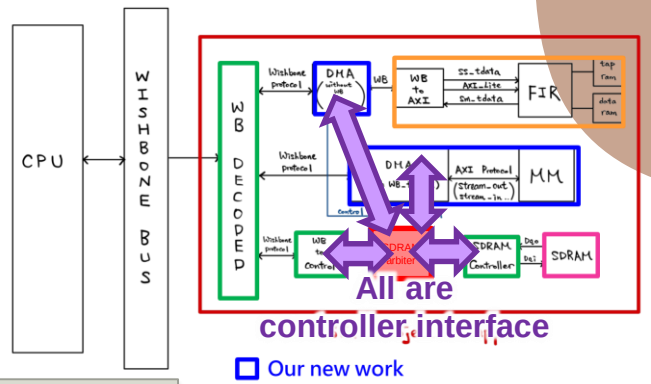
Our new work

Q2: Explain prefetch mechanism, combine with burst?

- SDRAM arbiter
  - Data在prefetch buffer中：直接return，不需request到SDRAM



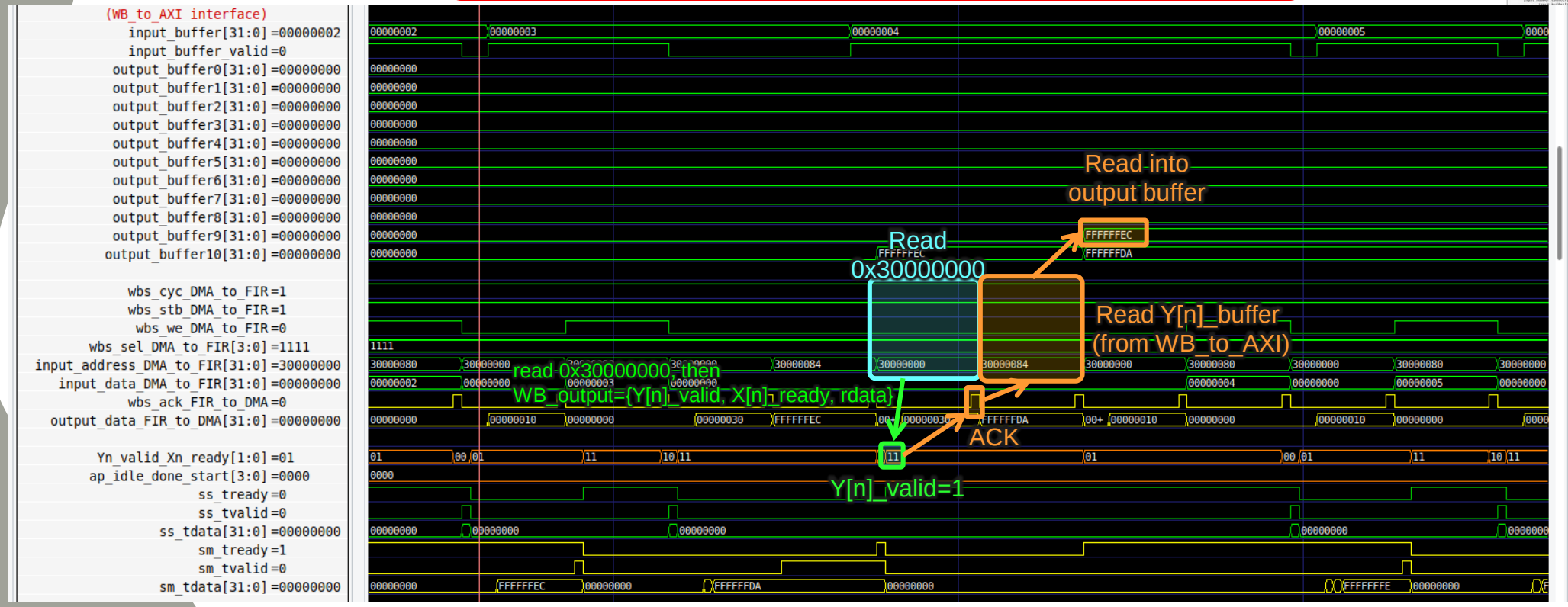


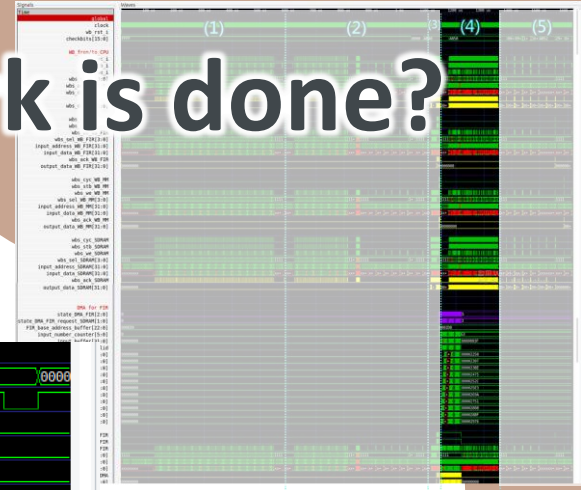


Q3: with DMA, how do you know the work is done?

- DMA for FIR : 2. When input buffer is valid, input to FIR

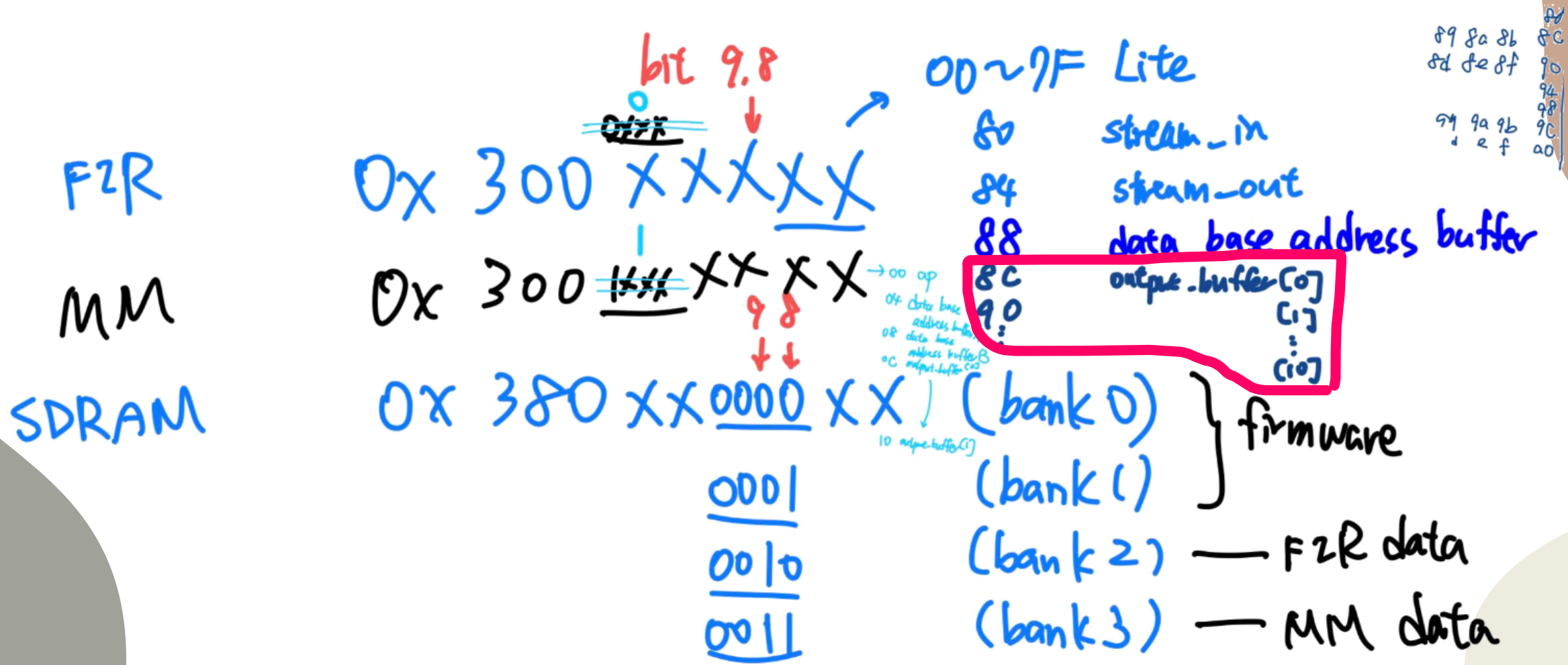
When Y[n] buffer is valid, output from FIR





Q3: with DMA, how do you know the work is done?

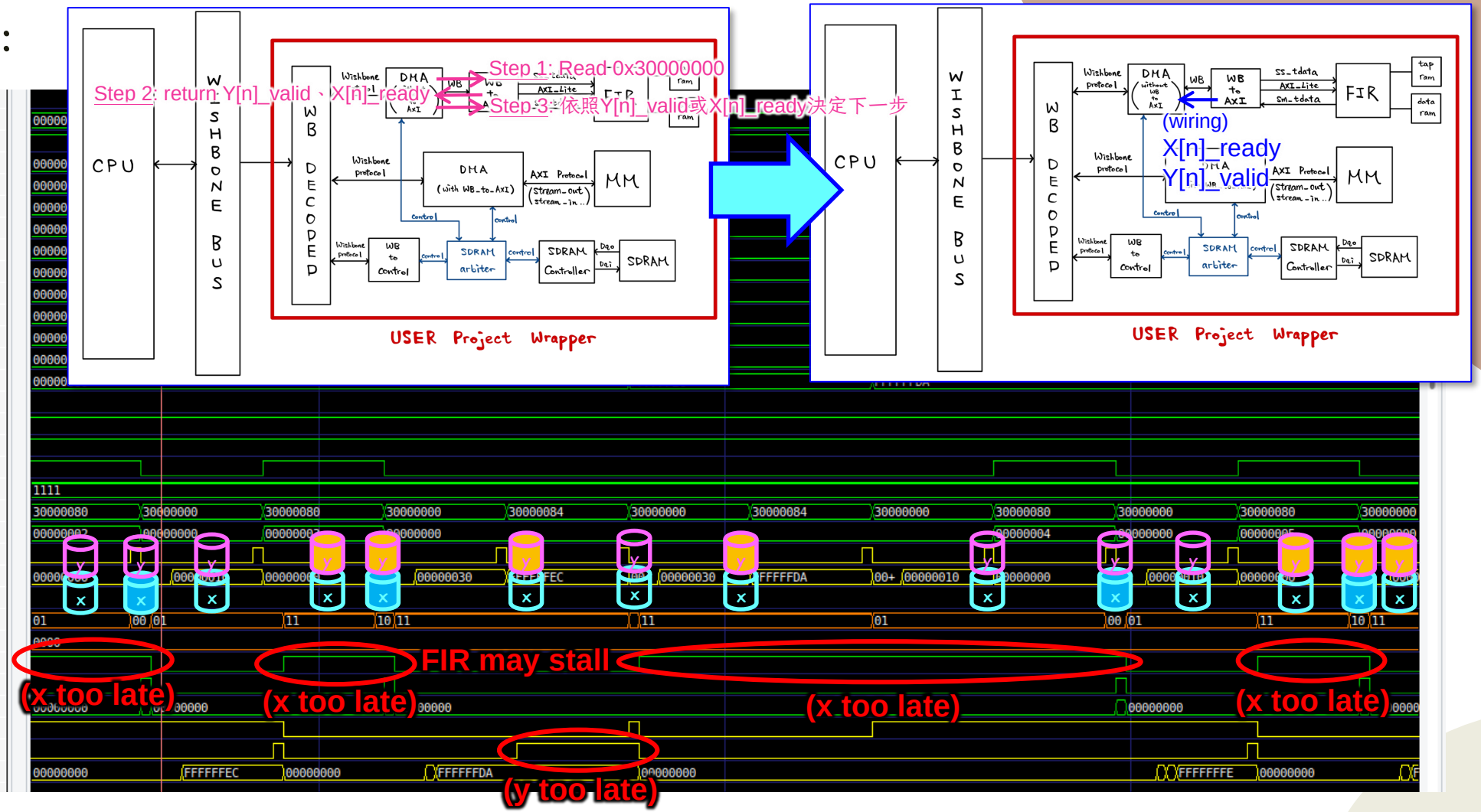
- Configuration address map：



Q4: What area you can further improve

- In DMA for FIR :

```
(WB_to_AXI interface)
input_buffer[31:0]=00000002
input_buffer_valid=0
output_buffer0[31:0]=00000000
output_buffer1[31:0]=00000000
output_buffer2[31:0]=00000000
output_buffer3[31:0]=00000000
output_buffer4[31:0]=00000000
output_buffer5[31:0]=00000000
output_buffer6[31:0]=00000000
output_buffer7[31:0]=00000000
output_buffer8[31:0]=00000000
output_buffer9[31:0]=00000000
output_buffer10[31:0]=00000000

wbs_cyc_DMA_to_FIR=1
wbs_stb_DMA_to_FIR=1
wbs_we_DMA_to_FIR=0
wbs_sel_DMA_to_FIR[3:0]=1111
input_address_DMA_to_FIR[31:0]=30000000
input_data_DMA_to_FIR[31:0]=00000000
wbs_ack_FIR_to_DMA=0
output_data_FIR_to_DMA[31:0]=00000000

Yn_valid_Xn_ready[1:0]=01
ap_idle_done_start[3:0]=0000
ss_tready=0
ss_tvalid=0
ss_tdata[31:0]=00000000
sm_tready=1
sm_tvalid=0
sm_tdata[31:0]=00000000
```

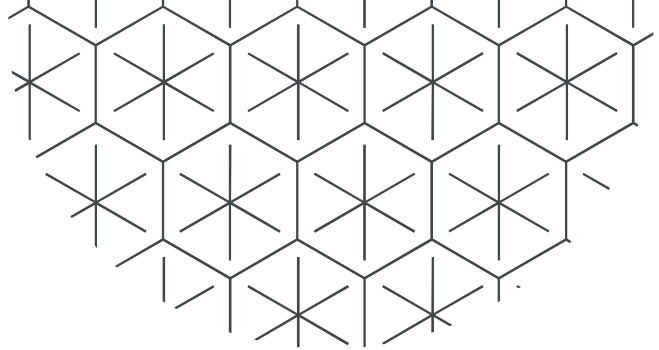


Non-ready x[n] buffer in WB_to_AXI module

Ready x[n] buffer in WB_to_AXI module

Invalid y[n] buffer in WB_to_AXI module

Valid y[n] buffer in WB_to_AXI module



Thank you!