Lab #D -SDRAM

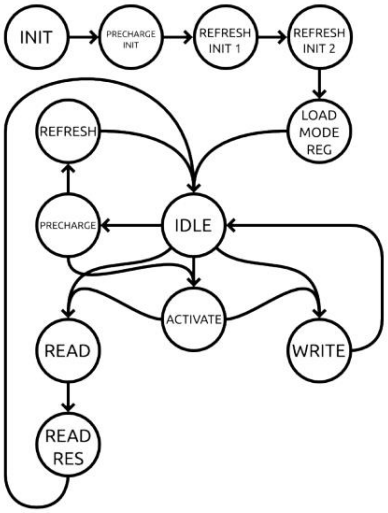
第九組

組員:紀浩洋、吳承哲、洪啓恩

SDRAM controller design

SDRAM Controller

- INIT→IDLE
- IDLE→ACTIVATE→WRITE→IDLE
- IDLE→ACTIVATE→READ→READ_RES →IDLE
- IDLE→WRITE→IDLE
- IDLE→READ→READ_RES→IDLE
- IDLE→PRECHARE→ACTIVATE→WRITE →IDLE
- IDLE→PRECHARE→ACTIVATE→READ →READ_RES→IDLE
- IDLE→PRECHARE→REFRESH→IDLE



Reference: https://alchitry.com/sdram-mojo

INIT : The initial state where the SDRAM is initialized, and the internal logic control unit is initialized before normal SDRAM operation. In this lab, the process will proceed directly to the IDLE stage.
INIT --> IDLE

IDLE : Idle state, where after a bank is precharged, the SDRAM will be in the IDLE stage after a certain period.
ACTIVATE (row activate): When in the IDLE state, upon receiving a control signal, the bank and row are selected, transitioning to the ACTIVATE state.
IDLE --> ACTIVATE

WRITE : In the row activate state, upon receiving a WRITE signal, the bank and column are selected, and the data writing process is executed.
IDLE --> ACTIVATE --> WRITE --> IDLE (Writing steps within the same bank)
IDLE --> PRECHARGE --> ACTIVATE --> WRITE --> IDLE (Writing steps involving different banks)

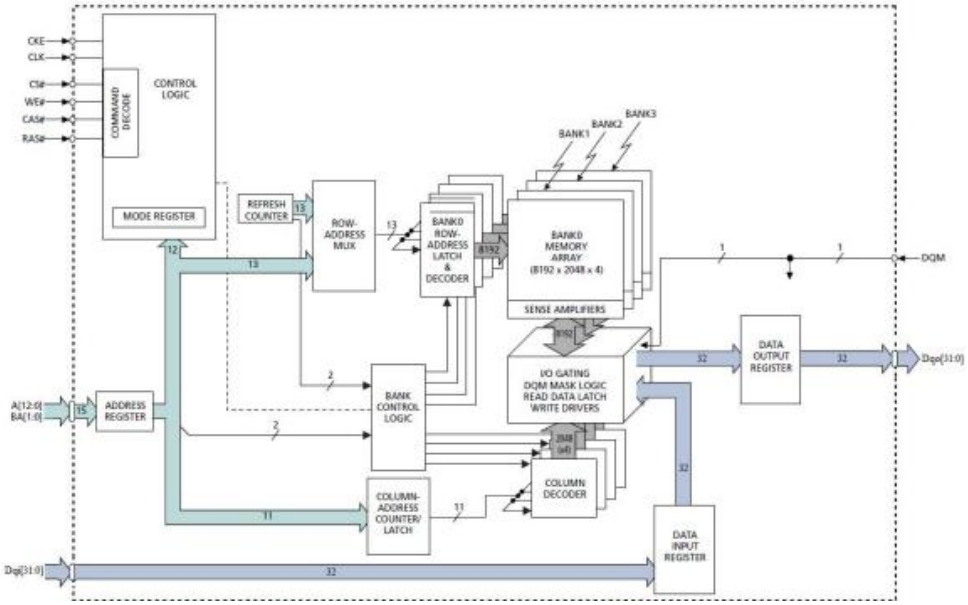
READ : In the row activate state, upon receiving a READ signal, the bank and column are selected, responsible for executing the SDRAM read steps.
IDLE --> ACTIVATE --> READ --> READ_RES --> IDLE (Reading steps within the same bank)
IDLE --> PRECHARGE --> ACTIVATE --> READ --> READ_RES --> IDLE (Reading steps involving different banks)

READ_RES : Reset performed after completing a read.

PRECHARGE : Precharging, necessary when closing a row. To perform read or write operations on a different bank, precharging is required to close the original bank and row.

REFRESH : To maintain the data in SDRAM, continuous refreshing is needed, mainly divided into two types: auto refresh and self-refresh. During auto refresh, all active banks need to undergo precharge before execution.

SDRAM bus protocol



clk: Clock signal

Cke (sdram_sle): Enable signal for the clk signal

Cas_n (sdram_cas): Column address strobe signal, input data for selecting the row address.

Ras_n (sdram_ras): Row address strobe signal, input data for selecting the column address.

We_n (sdram_we): Enable signal for write operations.

Dqm (sdram_dqm): Data mask (Data I/O Mask), used to control which input or output data ports are disabled.

Ba (sdram_ba): Bank address, primarily used to select which bank to use.

Addr (saram_a): Address bus, depending on the current situation, it may be the address of the row or column.

Dqi (sdram_dqi): Input for the original in-out pin.
Dqo (sdram_dqo): Output for the original in-out pin.

Introduce the prefetch scheme

```
/////////////// prefetch //////////////////
always @(posedge clk) begin
    if (in_valid) begin
        prefetch_address <= addr + 22'd4;
        prefetch <= (prefetch_address == addr) && !rw;
    end else if (!in_valid && out_valid_q) begin
        prefetch_address <= 0;
        prefetch <= 0;
    end else begin
        prefetch <= 0;
    end
end
```

This is the condition setting for prefetching. Initially, when in_valid equals 1, the prefetch_address is incremented by 4. Next, it determines when prefetch should be set to 1, and this decision is made in the IDLE state.

During the implementation process, initially, we placed the prefetch bank access in the READ_RES state and checked whether prefetching should be done in the IDLE state. Assuming prefetching is required, it would jump to the READ_RES state to access the prefetch address. While this approach allowed prefetching, if the second read was from the same BANK, it still required a re-ACTIVATE. Consequently, we later added prefetch bank access even in the IDLE state. This means that if the second read is from the same bank, there is no need for a re-ACTIVATE.

In other words, during the first READ, an ACTIVATE is necessary, and this part requires a waiting time of 3 cycles. In the second instance, if the read is from the same BANK, only 3 cycles of READ are needed to retrieve the data. As a result, the time spent in READ_RES is only 1 cycle, eliminating the need for the 3-cycle ACTIVATE waiting time.

Introduce the bank interleave for code and data

Separating Code and Data into different Banks facilitates the design of prefetching, providing clarity on which addresses store data to be loaded into the Processing Element (PE) for computation.

Bank Determination:

[1] The last 22 bits of the WB address are directly mapped to the SDRAM address.

Bank 0	3800_X 0 XX	3800_X 4 XX	3800_X 8 XX	3800_X C XX
Bank 1	3800_X 1 XX	3800_X 5 XX	3800_X 9 XX	3800_X D XX
Bank 2	3800_X 2 XX	3800_X 6 XX	3800_X A XX	3800_X E XX
Bank 3	3800_X 3 XX	3800_X 7 XX	3800_X B XX	3800_X F XX

Then, using wbs_adr_i[31:24], wbs_stb_i, and wbs_cyc_i to differentiate between req_code (3800) and req_data (3000). If request_code is 1, it indicates the need to look at code_adr and select the appropriate bank based on its data. The bank selection is either 2 (10) or 3 (11).
code_addr={wbs_adr_i[22:10], code_bank, wbs_adr_i[7:0]}

If req_data is 1, it indicates the need to look at the data_addr's address and choose which bank to use. The bank selection is either 0 (00) or 1 (01).
data_addr={wbs_adr_i[22:10], data_bank, wbs_adr_i[7:0]}

```
wire req_data;
wire req_code;
wire[1:0] data_bank;//data
wire [22:0] data_addr;
wire[1:0] code_bank;//code
wire [22:0] code_addr;
assign req_data = (wbs_stb_i & wbs_cyc_i &(wbs_adr_i[31:24] == 8'h30)) ? 1 : 0;
assign req_code = (wbs_stb_i & wbs_cyc_i &(wbs_adr_i[31:24] == 8'h38)) ? 1 : 0;
assign data_bank = (wbs_adr_i[9:8]>2'b01) ? (wbs_adr_i[9:8]-2'b10) : wbs_adr_i[9:8];
assign data_addr = (req_data) ? {wbs_adr_i[22:10], data_bank, wbs_adr_i[7:0]} : 0;
assign code_bank = (wbs_adr_i[9:8]<2'b10) ? (wbs_adr_i[9:8]+2'b10) : wbs_adr_i[9:8];
assign code_addr = (req_code) ? {wbs_adr_i[22:10], code_bank, wbs_adr_i[7:0]} : 0;
```

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

```
wire[1:0] data_bank;
assign data_bank = (wbs_adr_i[9:8]>2'b01) ? (wbs_adr_i[9:8]-2'b10) : wbs_adr_i[9:8];
wire [22:0] data_addr;
assign data_addr = (req_data) ? {wbs_adr_i[22:10], data_bank, wbs_adr_i[7:0]} : 0;

wire[1:0] code_bank;
assign code_bank = (wbs_adr_i[9:8]<2'b10) ? (wbs_adr_i[9:8]+2'b10) : wbs_adr_i[9:8];
wire [22:0] code_addr;
assign code_addr = (req_code) ? {wbs_adr_i[22:10], code_bank, wbs_adr_i[7:0]} : 0;
```

To set which bank to use within the user configuration, the following code is utilized. The decision is made based on the value of wbs_adr_i[9:8]. If wbs_adr_i[9:8] > 01, then data_bank is determined as wbs_adr_i[9:8] - 2'b10 (00 or 01). If wbs_adr_i[9:8] < 10, then code_bank is determined as wbs_adr_i[9:8] + 2'b10 (10 or 11).

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

The reliability of data in SDRAM is ensured through the use of the refresh technique, which involves periodic refreshing operations. However, conflicts may arise when system accesses and refresh operations occur simultaneously on the same section. To mitigate such situations, it is advisable to reduce the refresh cycle, minimizing the chances of conflicts.

Result: