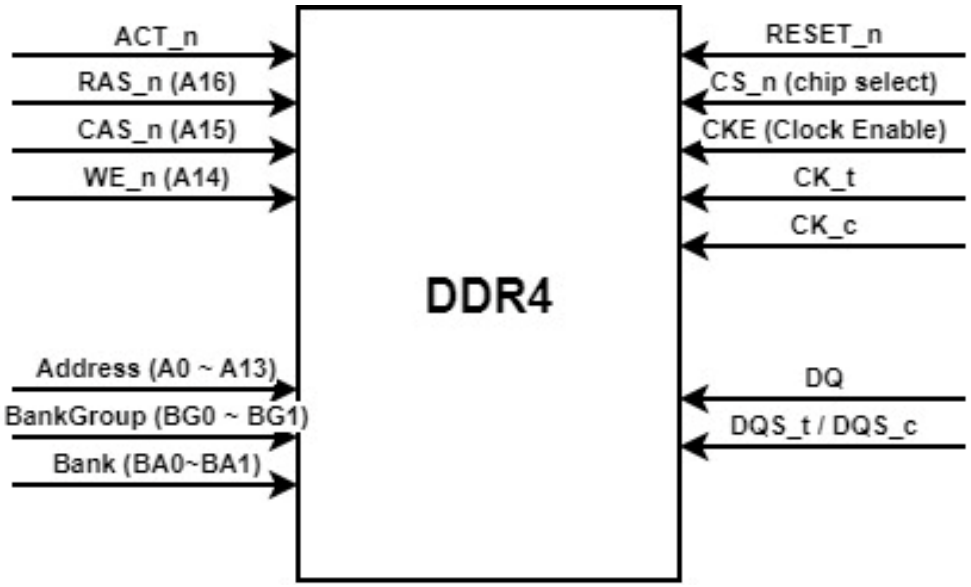
Ramulator 2.0 Summary

Intelligent System

Laboratory

DDR4 Summary

❑ DDR4 basic signals



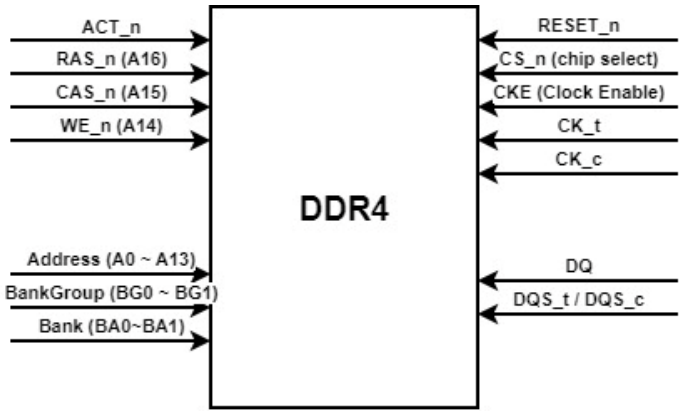
Symbol	Type	Function
RESET_n	Input	DRAM is active only when this signal is HIGH
CS_n	Input	The memory looks at all the other inputs only if this is LOW.
CKE	Input	Clock Enable. HIGH activates internal clock signals and device input buffers and output drivers.
CK_t/CK_c	Input	Differential clock inputs. All address & control signals are sampled at the crossing of posedge of CK_t & negedge of CK_n.
DQ/DQS	Inout	Data Bus & Data Strobe. This is how data is written in and read out. The strobe is essentially a data valid flag.
RAS_n/A16 CAS_n/A15 WE_n/A14	Input	These are dual function inputs. When ACT_n & CS_n are LOW, these are interpreted as Row Address Bits. When ACT_n is HIGH, these are interpreted as command pins to indicate READ, WRITE or other commands.
ACT_n	Input	Activate command input
BG0-1 BA0-1	Input	Bank Group, Bank Address
A0-13	Input	Address inputs

- **RAS_n / A16, CAS_n / A15, WE_n / A14 : Dual Function Inputs**
  - **이중 기능 입력. ACT_n과 CS_n이 LOW일 때 행 주소 bit로 해석**
  - **ACT_n이 HIGH일 때 READ, WRITE 또는 다른 command을 나타내는 command pin으로 해석**

Reference: https://www.systemverilog.io/design/ddr4-basics/
Reference: https://computing-jhson.tistory.com/28#google_vignette

DDR4 Summary

❑ DDR4 basic signals



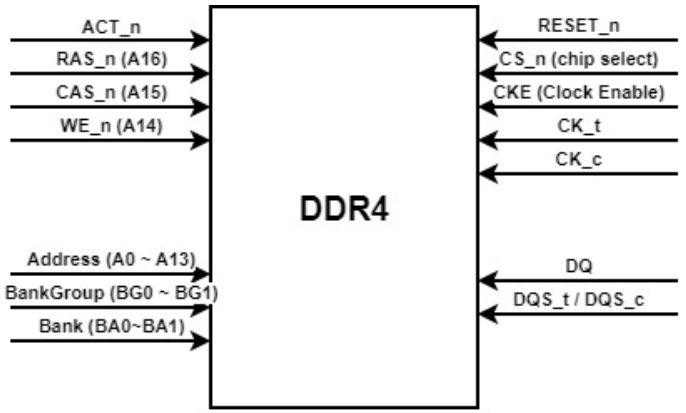
READ 시나리오

1. Issue Activate (ACT) command
   - Bank의 특정 row를 open (Data를 Sense Amp로 load)
   - Signals:
     - → CS_n: LOW
     - → ACT_n: LOW
     - → RAS_n/A16, CAS_n/A15, WE_n/A14: row address bit로 사용 (ex. A16=0, A15=0, A14=0)
     - → A0-13: row address 지정 (ex. row 0 = 0000...0).
     - → BG0-1, BA0-1: Bank Group/Band 선택 (ex. BG=00, BA=00)
     - → CKE: HIGH
   - Timing Constraints:
     - → ACT command 후 tRCD(Row to Column Delay) 대기
   - 내부 동작:
     - → Row Decoder가 row를 select, Bit Line을 통해 data가 sense amp로 이동 (row opened)
     - → Data는 아직 DQ pin로 나오지 않음
2. Issue Read (RD) command
3. Data Burst
4. Issue Precharge (PRE) command

Reference: https://www.systemverilog.io/design/ddr4-basics/
Reference: https://computing-jhson.tistory.com/28#google_vignette

DDR4 Summary

❑ DDR4 basic signals



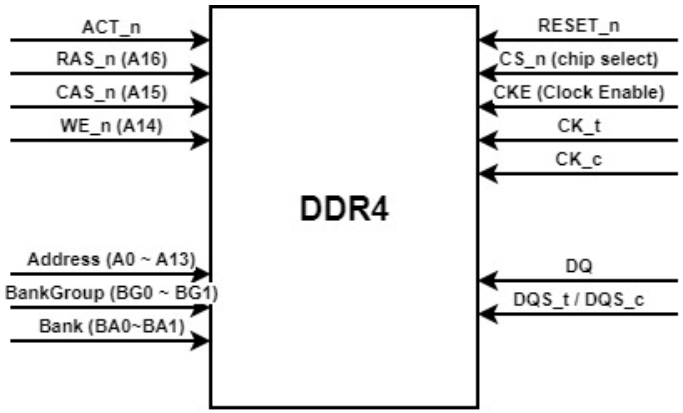
READ 시나리오

1. Issue Activate (ACT) command
2. Issue Read (RD) command
   - Column을 선택해 Sense Amp의 Data를 DQ bus로 출력
   - Signals:
     → CS_n: LOW
     → ACT_n: HIGH
     → RAS_n/A16, CAS_n/A15, WE_n/A14: Column Address Strobe로 사용 (ex. A16=H, A15=L, A14=H)
     → A0-13: column address 지정 (ex. column 0 = 0000...0)
     → BG0-1, BA0-1: 동일 뱅크 유지(ex. BG=00, BA=00)
     → CKE: HIGH
   - Timing Constraints:
     → RD Command 후 tCL(CAS Latency) 대기
   - 내부 동작:
     → Column Decoder가 column을 select, data가 I/O buffer로 이동
3. Data Burst
4. Issue Precharge (PRE) command

Reference: https://www.systemverilog.io/design/ddr4-basics/
Reference: https://computing-jhson.tistory.com/28#google_vignette

DDR4 Summary

❑ DDR4 basic signals



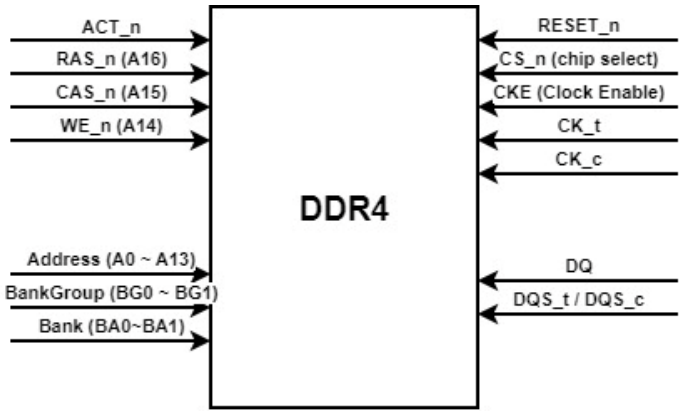
READ 시나리오

1. Issue Activate (ACT) command
2. Issue Read (RD) command
3. Data Burst
   - 실제 Data 전송
   - Signals:
     → DQ: Data Bus (8bit x 8 burst = 64bit data 출력)
     → DQS: Data Strobe (CLK처럼 Edge에서 Data Valid 표시)
     Read 시 메모리가 DQS를 생성 (source synchronous)
   - Timing:
     → DQS의 rising/falling edge에서 DQ data Capture (DDR: Double Data Rate)
   - 내부 동작:
     → burst 길이만큼 (BL=8) data가 연속 출력
     → Memory Ctrlr가 DQS를 사용해 Data Capture
4. Issue Precharge (PRE) command

Reference: https://www.systemverilog.io/design/ddr4-basics/
Reference: https://computing-jhson.tistory.com/28#google_vignette

DDR4 Summary

❑ DDR4 basic signals



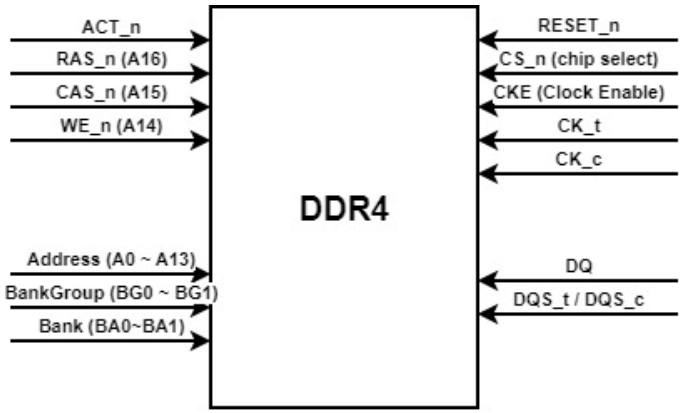
READ 시나리오

1. Issue Activate (ACT) command
2. Issue Read (RD) command
3. Data Burst
4. Issue Precharge (PRE) command
   - row를 닫아 해당 Bank를 초기화
   - Signals:
     → CS_n: LOW
     → ACT_n: HIGH
     → RAS_n/A16, CAS_n/A15, WE_n/A14: Row Address Strobe로 사용 (ex. A16=L, A15=H, A14=H)
     → A10: 1 (All banks precharge) 또는 0 (Single bank)
     → BG0-1, BA0-1: 대상 뱅크
     → CKE: HIGH
   - Timing Constraints:
     → tRP (Row Precharge time) 후 다음 ACT 가능
   - 내부 동작:
     → row 닫음 후 Sense Amp Data 복원 (Precharge: VDD/2)

Reference: https://www.systemverilog.io/design/ddr4-basics/
Reference: https://computing-jhson.tistory.com/28#google_vignette

DDR4 Summary

❑ DDR4 basic signals



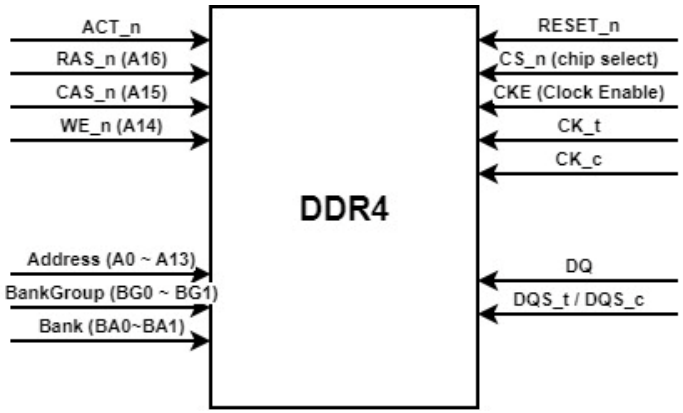
WRITE 시나리오

1. Issue Activate (ACT) command
   - Bank의 특정 row를 open (Data를 Sense Amp로 load)
   - Signals:
     - → CS_n: LOW
     - → ACT_n: LOW
     - → RAS_n/A16, CAS_n/A15, WE_n/A14: row address bit로 사용 (ex. A16=0, A15=0, A14=0)
     - → A0-13: row address 지정 (ex. row 0 = 0000...0).
     - → BG0-1, BA0-1: Bank Group/Band 선택 (ex. BG=00, BA=00)
     - → CKE: HIGH
   - Timing Constraints:
     - → ACT command 후 tRCD(Row to Column Delay) 대기
   - 내부 동작:
     - → Row Decoder가 row를 select, Bit Line을 통해 data가 sense amp로 이동 (row opened)
     - → Data는 아직 DQ pin로 나오지 않음
2. Issue Read (RD) command
3. Data Burst
4. Issue Precharge (PRE) command

Reference: https://www.systemverilog.io/design/ddr4-basics/
Reference: https://computing-jhson.tistory.com/28#google_vignette

DDR4 Summary

❑ DDR4 basic signals



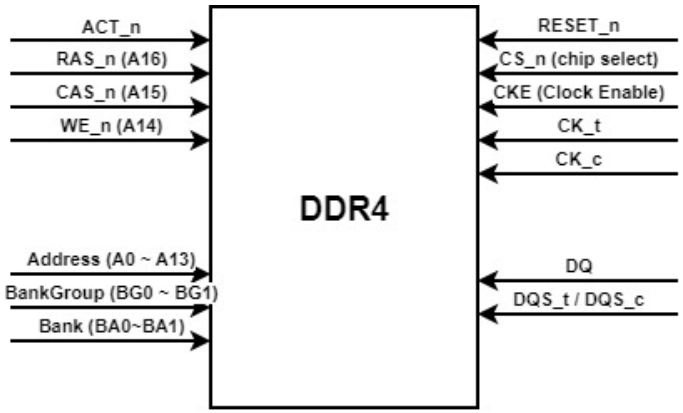
WRITE 시나리오

1. Issue Activate (ACT) command
2. Issue Write (WR) command
   - Column을 선택해 DQ bus의 Data를 Sense Amp를 통해 Write
   - Signals:
     - → CS_n: LOW
     - → ACT_n: HIGH
     - → RAS_n/A16, CAS_n/A15, WE_n/A14: Column Address Strobe로 사용 (ex. A16=H, A15=L, A14=L)
     - → A0-13: column address 지정 (ex. column 0 = 0000...0)
     - → BG0-1, BA0-1: 동일 뱅크 유지(ex. BG=00, BA=00)
     - → CKE: HIGH
   - Timing Constraints:
     - → WR command 후 tCWL(CAS Write Latency) 대기
   - 내부 동작:
     - → Column Decoder가 column을 select, write 준비
3. Data Burst
4. Issue Precharge (PRE) command

Reference: https://www.systemverilog.io/design/ddr4-basics/
Reference: https://computing-jhson.tistory.com/28#google_vignette

DDR4 Summary

❑ DDR4 basic signals



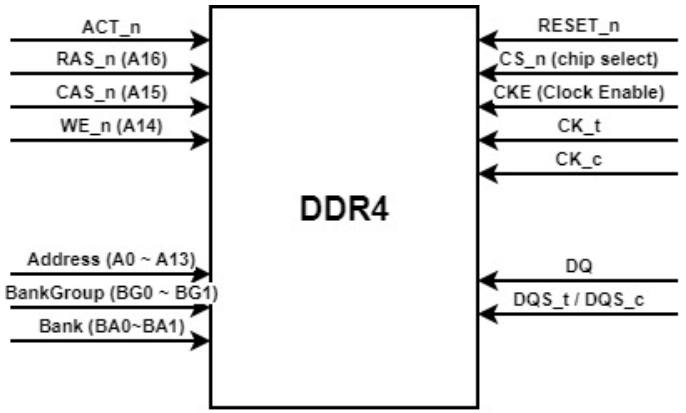
WRITE 시나리오

1. Issue Activate (ACT) command
2. Issue Read (RD) command
3. Data Burst
   - 실제 Data 전송
   - Signals:
     → DQ: Data Bus (8bit x 8 burst = 64bit data 입력)
     → DQS: Data Strobe (CLK처럼 Edge에서 Data Valid 표시)
     Write 시 Controller가 DQS를 생성 (source synchronous)
   - Timing:
     → DQS의 rising/falling edge에서 DQ data Capture (DDR: Double Data Rate)
   - 내부 동작:
     → burst 길이만큼 (BL=8) data가 연속 입력
     → Memory가 DQS를 사용해 Data Capture
4. Issue Precharge (PRE) command

Reference: https://www.systemverilog.io/design/ddr4-basics/
Reference: https://computing-jhson.tistory.com/28#google_vignette

DDR4 Summary

❑ DDR4 basic signals



WRITE 시나리오

1. Issue Activate (ACT) command
2. Issue Read (RD) command
3. Data Burst
4. Issue Precharge (PRE) command
   - row를 닫아 해당 Bank를 초기화
   - Signals:
     → CS_n: LOW
     → ACT_n: HIGH
     → RAS_n/A16, CAS_n/A15, WE_n/A14: Row Address Strobe로 사용 (ex. A16=L, A15=H, A14=H)
     → A10: 1 (All banks precharge) 또는 0 (Single bank)
     → BG0-1, BA0-1: 대상 뱅크
     → CKE: HIGH
   - Timing Constraints:
     → tRP(Row Precharge time) 후 다음 ACT 가능
   - 내부 동작:
     → row 닫음 후 Sense Amp Data 복원 (Precharge: VDD/2)

Reference: https://www.systemverilog.io/design/ddr4-basics/
Reference: https://computing-jhson.tistory.com/28#google_vignette

DDR4 Summary

❑ DDR4 Hierarchy – Overview

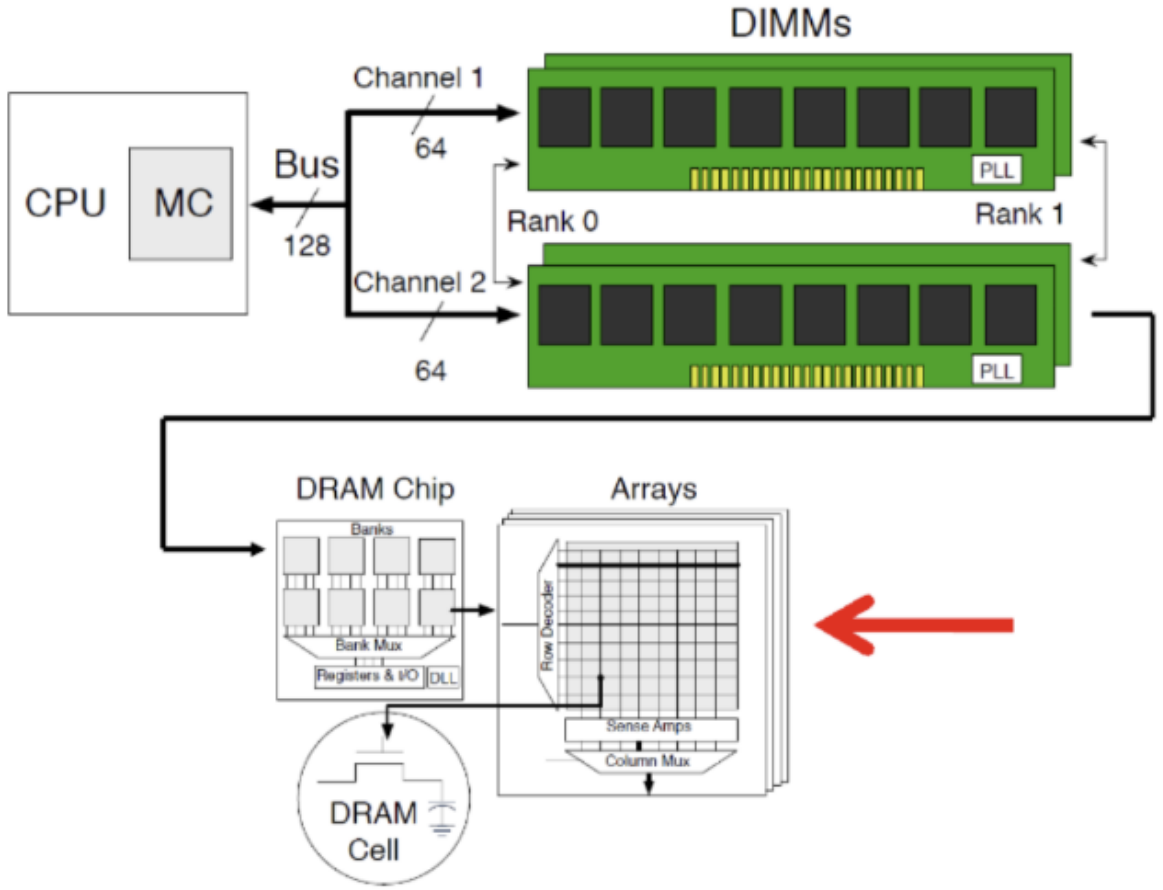


Fig 3. Main memory system 요약 [1]

Reference: https://www.systemverilog.io/design/ddr4-basics/
Reference: https://computing-jhson.tistory.com/28#google_vignette

DDR4 Summary

❑ DDR4 Hierarchy - Channel level

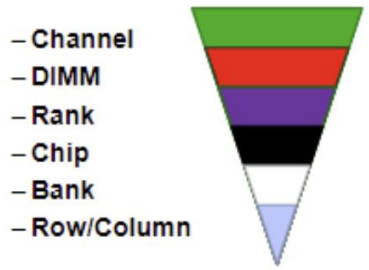


Fig 2. Memory subsystem organization [1]

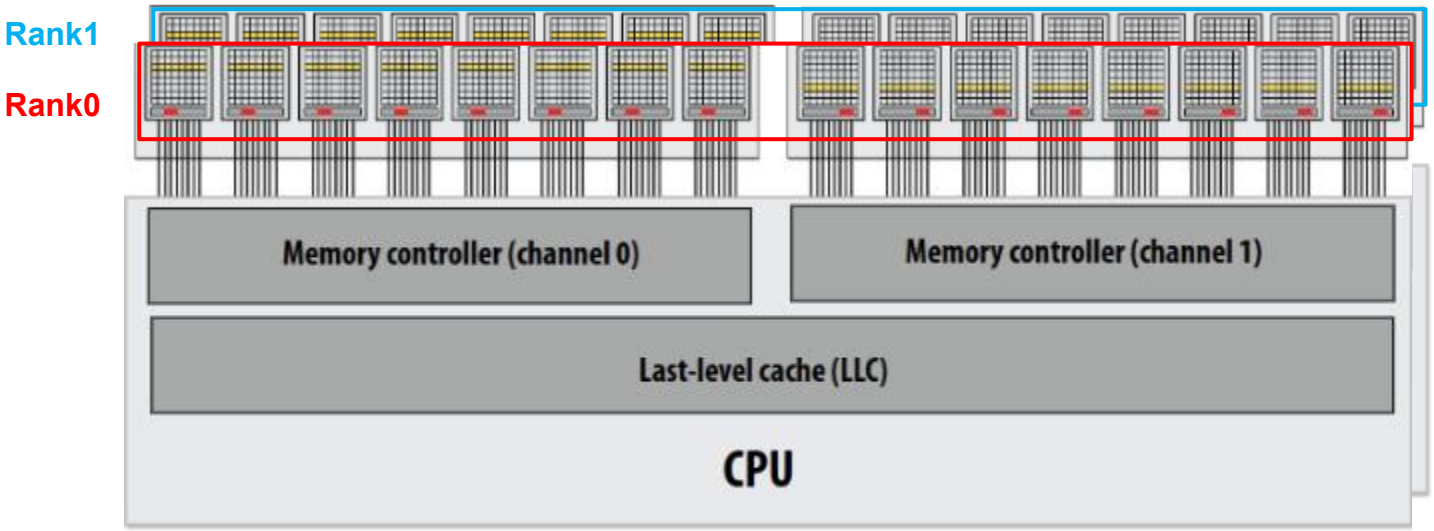


Fig 6. DRAM multi-channel 구조 [2]

- DRAM Channel는 각각 독립적인 Memory Controller에 할당됨
- 한 channel에 여러 개의 DIMM이 꽂혀 있으며, 1개의 DIMM도 여러 개의 rank(DIMM의 앞, 뒤)들로 구성
- 여러 Rank는 Data Bus를 공유 → CAS to CAS latency (timing Constraints 유발)

Reference: https://www.systemverilog.io/design/ddr4-basics/
Reference: https://computing-jhson.tistory.com/28#google_vignette

DDR4 Summary

❑ DDR4 Hierarchy - Rank level

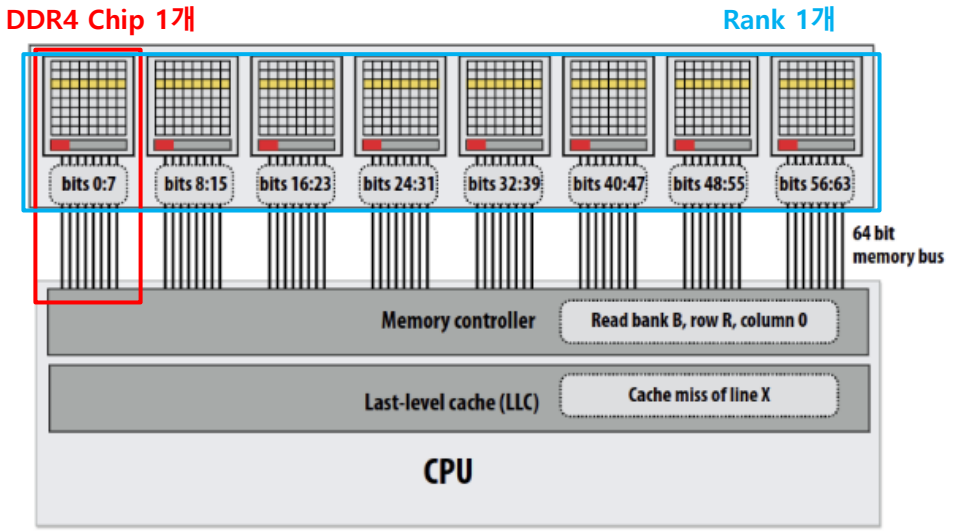
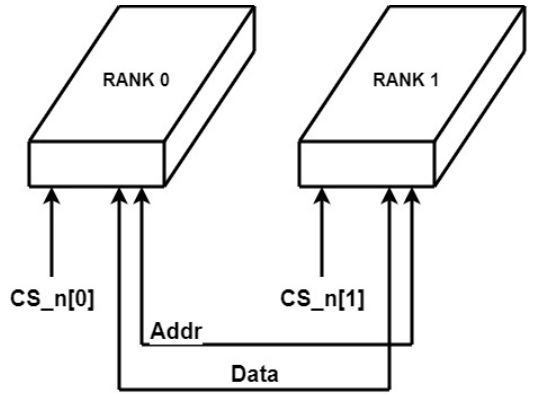


Fig 5. DRAM multi-chip 구조 [2]



DRAM Rank는 여러 Chip을 가지는 Level

- 이 여러 Chip들을 논리적으로 묶은 단위가 Rank
- Rank 1개당 1개의 Chip Select 신호가 할당

오른쪽 위의 그림은 64bit Transfer하는 Rank

- 8개의 chip(x8)으로 구성
- 해당 Rank는 1번의 memory operation에 64bit를 전송

오른쪽 아래 그림은 Dual Rank를 보여줌

- 2개의 Device는 동일한 Addr 및 Data Bus에 연결
- 각 Device를 별도로 지정하기 위해 2개의 ChipSelect가 필요
- 2개의 ChipSelect가 필요하므로, 이 설정을 Dual-Rank라고 함

Reference: https://www.systemverilog.io/design/ddr4-basics/
Reference: https://computing-jhson.tistory.com/28#google_vignette

DDR4 Summary

❑ DDR4 Hierarchy – Bank Group Level

1개 Chip 안에 여러 Bank Group

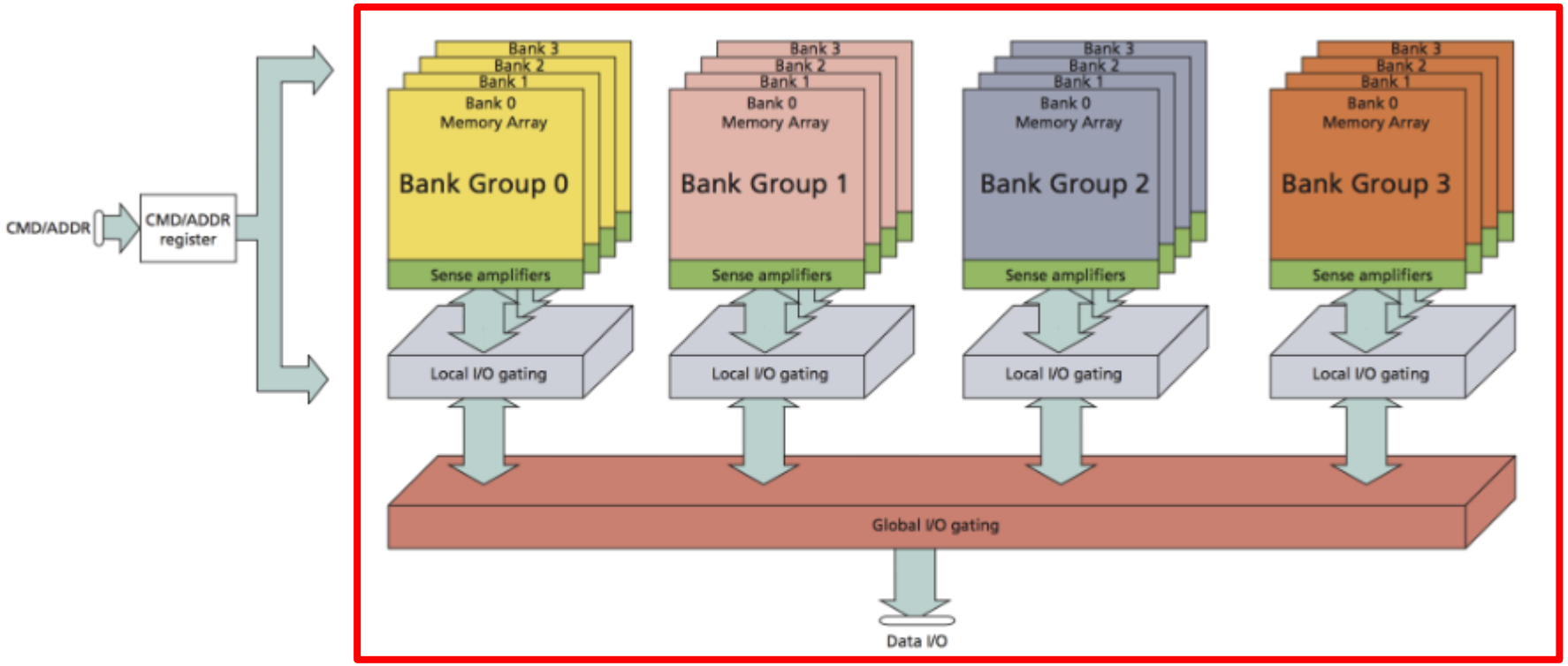


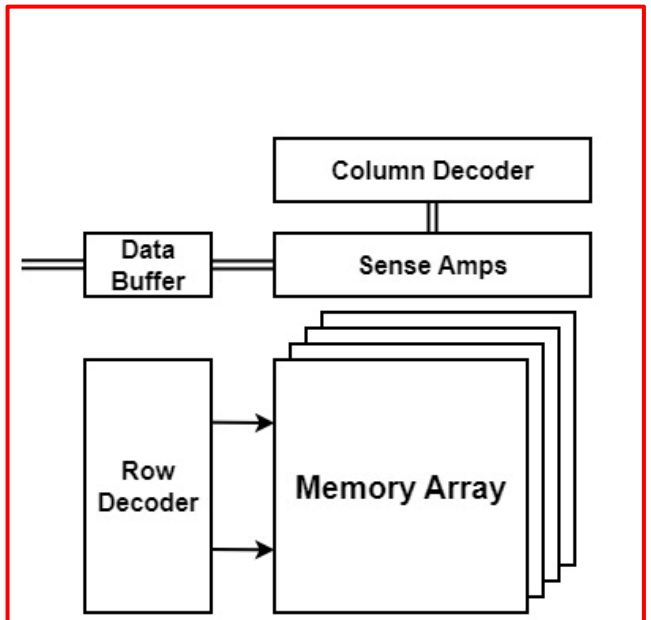
Figure 2: BankGroup & Bank (Source: Micron Datasheet)

Reference: https://www.systemverilog.io/design/ddr4-basics/
Reference: BlockHammer_preventing-DRAM-rowhammer-at-low-cost_hpca21

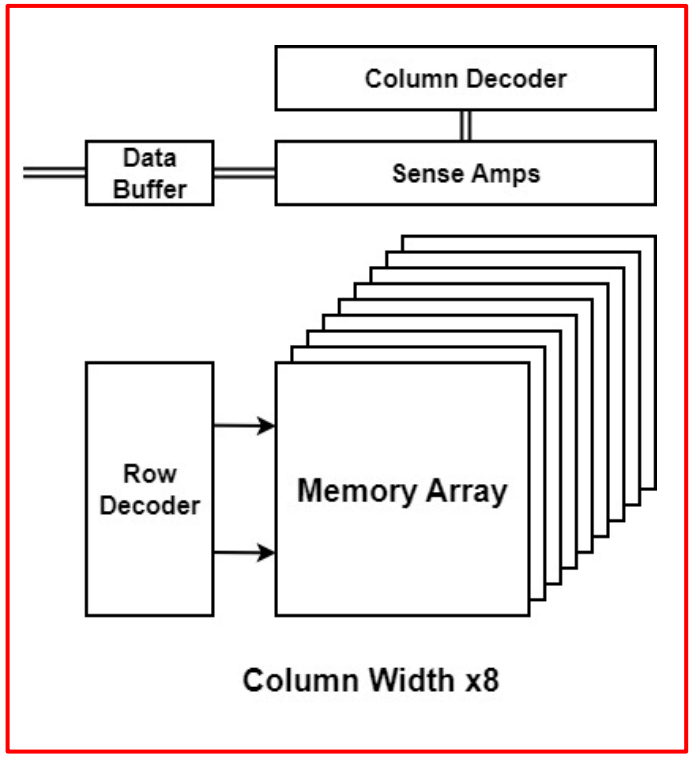
DDR4 Summary

❑ DDR4 Hierarchy – Each Bank (Bank Level)

Bank 1개



Bank 1개



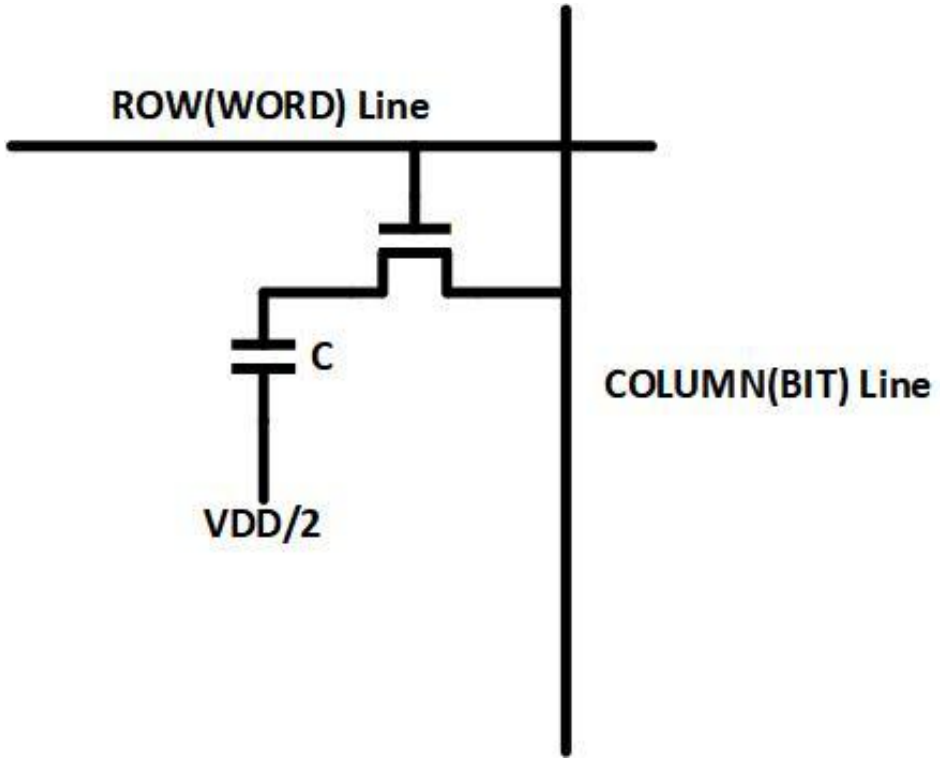
Column Address

- Sense Amp로 load된 Word의 일부를 Read
- Column의 Width = Bit Line
- Column Width은 표준
  - 4bit, 8bit 또는 16bit Width
  - DRAM은 이에 따라 x4, x8, x16으로 분류
  - DQ Data Bus Width = Column Width
    - x4 → 4 data pins DQ[3:0]
    - x8 → 8 data pins DQ[7:0]

Reference: https://www.systemverilog.io/design/ddr4-basics/
Reference: BlockHammer_preventing-DRAM-rowhammer-at-low-cost_hpca21

Bank Group, Bank, Row, Column

❑ DDR4 Hierarchy – CELL Level



Word Line ON

- Access TR이 ON되고, Bit Line의 Data가 들어옴

Bit Line Data

- Access TR이 ON이 되면 Bit Line을 통해 Read/Write Data가 Transfer됨

Refresh

- Capacitor는 Leaky (Dynamic)
- 주기적으로 Refresh 필요

Reference: https://www.systemverilog.io/design/ddr4-basics/
Reference: BlockHammer_preventing-DRAM-rowhammer-at-low-cost_hpca21

DDR4 Summary

❑ DRAM Sizing & Addressing

2 Gb Addressing Table

Configuration		512 Mb x4	256 Mb x8	128 Mb x16
Bank Address	# of Bank Groups	4	4	2
	BG Address	BG0~BG1	BG0~BG1	BG0
	Bank Address in a BG	BA0~BA1	BA0~BA1	BA0~BA1
Row Address		A0~A14	A0~A13	A0~A13
Column Address		A0~A9	A0~A9	A0~A9
Page size		512B	1KB	2KB

4 Gb Addressing Table

Configuration		1 Gb x4	512 Mb x8	256 Mb x16
Bank Address	# of Bank Groups	4	4	2
	BG Address	BG0~BG1	BG0~BG1	BG0
	Bank Address in a BG	BA0~BA1	BA0~BA1	BA0~BA1
Row Address		A0~A15	A0~A14	A0~A14
Column Address		A0~A9	A0~A9	A0~A9
Page size		512B	1KB	2KB

8 Gb Addressing Table

Configuration		2 Gb x4	1 Gb x8	512 Mb x16
Bank Address	# of Bank Groups	4	4	2
	BG Address	BG0~BG1	BG0~BG1	BG0
	Bank Address in a BG	BA0~BA1	BA0~BA1	BA0~BA1
Row Address		A0~A16	A0~A15	A0~A15
Column Address		A0~A9	A0~A9	A0~A9
Page size		512B	1KB	2KB

16 Gb Addressing Table

Configuration		4 Gb x4	2 Gb x8	1 Gb x16
Bank Address	# of Bank Groups	4	4	2
	BG Address	BG0~BG1	BG0~BG1	BG0
	Bank Address in a BG	BA0~BA1	BA0~BA1	BA0~BA1
Row Address		A0~A17	A0~A16	A0~A16
Column Address		A0~A9	A0~A9	A0~A9
Page size		512B	1KB	2KB

Figure 5: Addressing (Source : JESD79-4B Spec)

DRAM은 표준 크기로 제공
- DDR4: 2Gb, 4Gb, 8Gb 및 16Gb가 제공됨

Row Address
- 2Gb (x4): A0 ~ A14 → 총 15bit 사용 [cf. x4는 DQ pin의 width]
- 8Gb (x4): A0 ~ A16 → 총 17bit 사용

Column Address
- DRAM size와 관계없이 only 10 column bits 가짐: A0 to A9

DRAM Width
- DDR4에서 3 widths가 available : x4, x8, x16
- x4 : 4 data pins DQ[3:0]
- x8 : 8 data pins DQ[7:0]
- x16 : 16 data pins DQ[15:0]

보통 DRAM은 burst 8이 적용
- x4는 1번 RD command당 32bit 전송
- x8는 1번 RD command당 64bit 전송

Reference: https://www.systemverilog.io/design/ddr4-basics/
Reference: BlockHammer_preventing-DRAM-rowhammer-at-low-cost_hpca21

DDR4 Summary

❑ DRAM Page

2 Gb Addressing Table

Configuration		512 Mb x4	256 Mb x8	128 Mb x16
Bank Address	# of Bank Groups	4	4	2
	BG Address	BG0~BG1	BG0~BG1	BG0
	Bank Address in a BG	BA0~BA1	BA0~BA1	BA0~BA1
Row Address		A0~A14	A0~A13	A0~A13
Column Address		A0~A9	A0~A9	A0~A9
Page size		512B	1KB	2KB

4 Gb Addressing Table

Configuration		1 Gb x4	512 Mb x8	256 Mb x16
Bank Address	# of Bank Groups	4	4	2
	BG Address	BG0~BG1	BG0~BG1	BG0
	Bank Address in a BG	BA0~BA1	BA0~BA1	BA0~BA1
Row Address		A0~A15	A0~A14	A0~A14
Column Address		A0~A9	A0~A9	A0~A9
Page size		512B	1KB	2KB

8 Gb Addressing Table

Configuration		2 Gb x4	1 Gb x8	512 Mb x16
Bank Address	# of Bank Groups	4	4	2
	BG Address	BG0~BG1	BG0~BG1	BG0
	Bank Address in a BG	BA0~BA1	BA0~BA1	BA0~BA1
Row Address		A0~A16	A0~A15	A0~A15
Column Address		A0~A9	A0~A9	A0~A9
Page size		512B	1KB	2KB

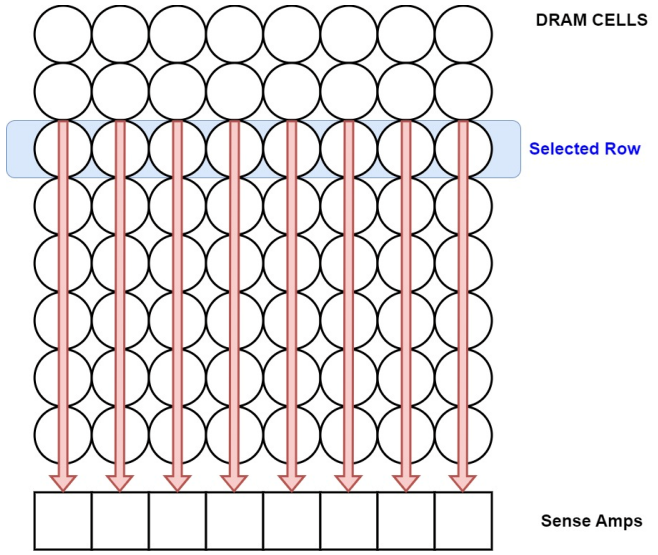
16 Gb Addressing Table

Configuration		4 Gb x4	2 Gb x8	1 Gb x16
Bank Address	# of Bank Groups	4	4	2
	BG Address	BG0~BG1	BG0~BG1	BG0
	Bank Address in a BG	BA0~BA1	BA0~BA1	BA0~BA1
Row Address		A0~A17	A0~A16	A0~A16
Column Address		A0~A9	A0~A9	A0~A9
Page size		512B	1KB	2KB

Figure 5: Addressing (Source : JESD79-4B Spec)

DRAM Page는 Row를 Activate할 때의 Sense Amp로 load되는 Bit 수

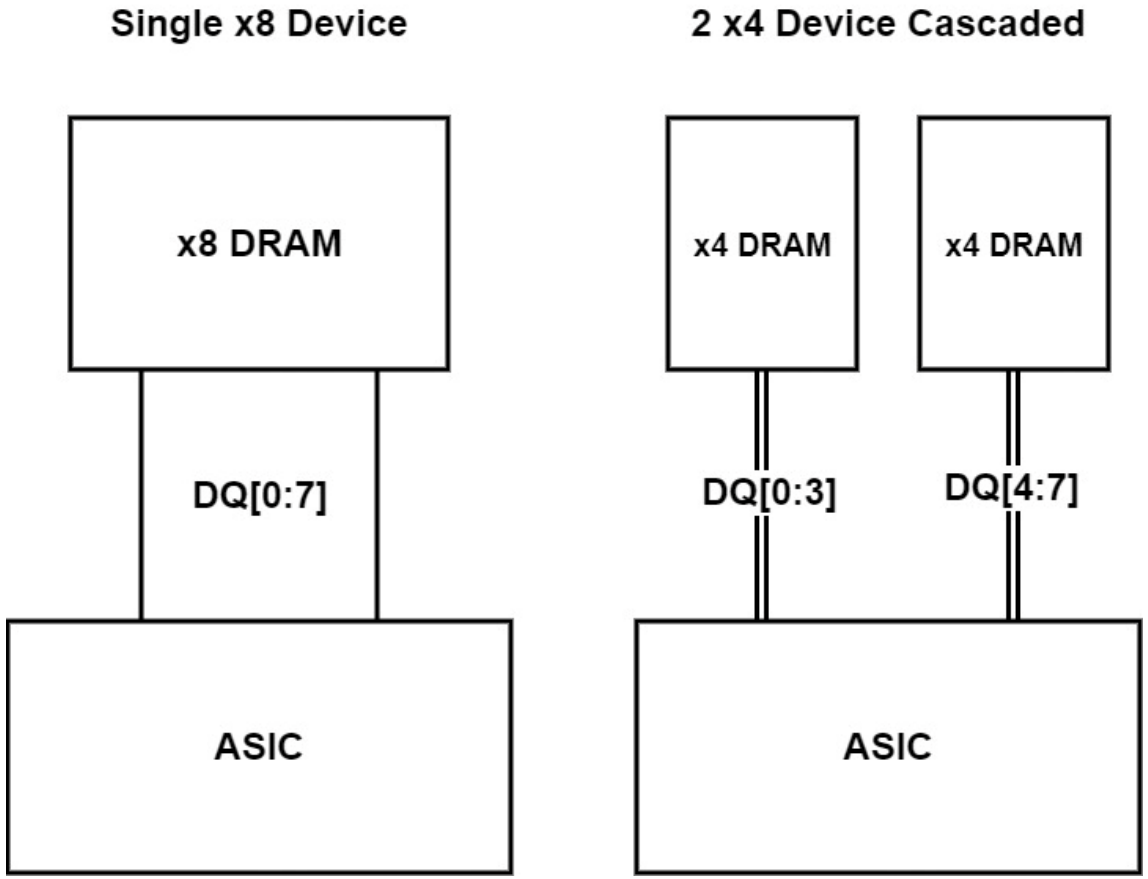
- Column Address is 10 bits → 1K bit-lines per row
- x4 device → 4K bit-lines per row = 512B per row
- x8 device → 8K bit-lines per row = 1KB per row
- x16 device → 16K bit-lines per row = 2KB per row



Reference: https://www.systemverilog.io/design/ddr4-basics/
Reference: BlockHammer_preventing-DRAM-rowhammer-at-low-cost_hpca21

DDR4 Summary

❑ DRAM Width Cascaded



DRAM Rank는 Bank Group을 내부에 가지는 Level

- Rank 1개당 1개의 Chip Select 신호가 할당

오른쪽 그림은 Dual Rank를 보여줌

- 2개의 Device는 동일한 Addr 및 Data Bus에 연결
- 각 Device를 별도로 지정하기 위해 2개의 ChipSelect가 필요
- 2개의 ChipSelect가 필요하므로, 이 설정을 Dual-Rank라고 함

DDR4 Summary

❑ Accessing Memory

RD 및 WR command → 2단계 Process

1. ACT command
   - ACT command와 동시에 전달된 Addr bit를 이용하여
     → BankGroup, Bank 및 Row을 선택 (RAS - Row Address Strobe)

2. RD or WR command
   - RD or WR command와 동시에 전달된 Addr bit를 이용하여
     → Burst Transfer의 시작 Column 위치를 선택 (CAS - Column Address Strobe)

3. 각 Bank에는 하나의 Sense Amp Set가 존재
   - 동일한 Bank에서 다른 Row에 대한 RD/WR를 수행하기 전
     → 현재 열린 행은 PRE Command을 사용하여 비활성화해야 함
     → PRE Command Issue 시, Sense Amp의 Data를 Row에 있는 CELL로 다시 복구 및 Row Close
     → Row Close 후 Bit Line을 VDD/2로 Precharge

Cf. Row를 비활성화하기 위해 RDA (Read with Auto-Precharge) or WRA command 사용 가능
   - Column Addr가 Addr bit A0-A9만 사용하므로, CAS 동안 사용되지 않는 A10은 Auto-Precharge를 나타냄

Reference: https://www.systemverilog.io/design/ddr4-basics/
Reference: BlockHammer_preventing-DRAM-rowhammer-at-low-cost_hpca21

DDR4 Summary

❑ DRAM Commands

- DRAM은 아래 Truth Table에 기반하여 ACT_n, RAS_n, CAS_n 및 WE_n 입력의 조합을 Command로 해석

Function	Shortcode	CS_n	ACT_n	RAS_n/A16	CAS_n/A15	WE_n/A14	A10/AP
Refresh	REF	L	H	L	L	H	H or L
Single Bank Precharge	PRE	L	H	L	H	L	L
Bank Activate	ACT	L	L	Row Address			
Write	WR	L	H	H	L	L	L
Write with Auto-Precharge	WRA	L	H	H	L	L	H
Read	RD	L	H	H	L	H	L
Read with Auto-Precharge	RDA	L	H	H	L	H	H

Basic DRAM Commands

Partial Command Truth-Table

Reference: https://www.systemverilog.io/design/ddr4-basics/
Reference: BlockHammer_preventing-DRAM-rowhammer-at-low-cost_hpca21

DDR4 Summary

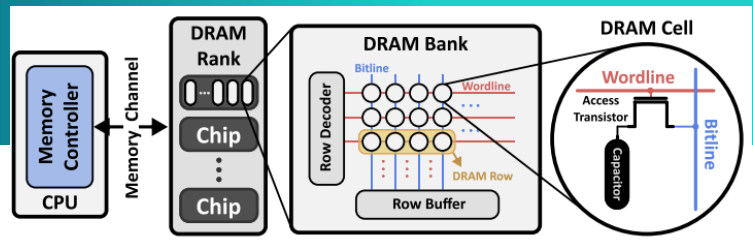
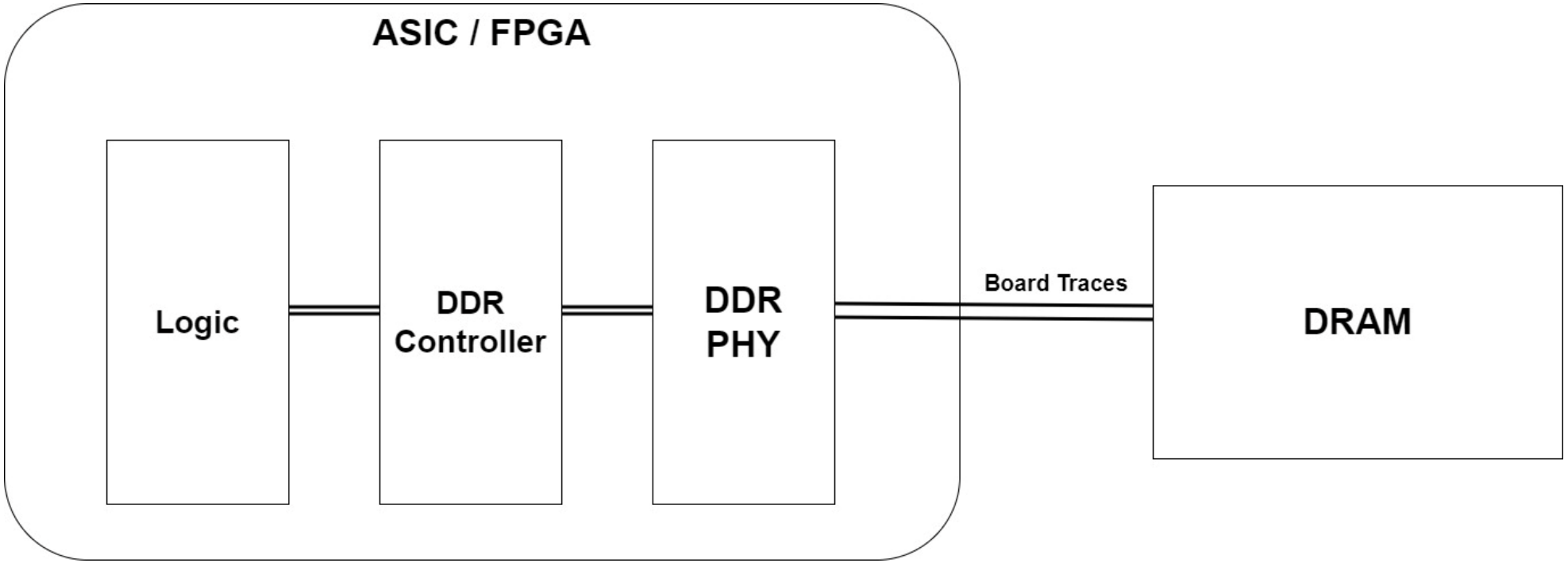


Figure 1: Structure of a typical DRAM-based system.

❑ DRAM sub-system

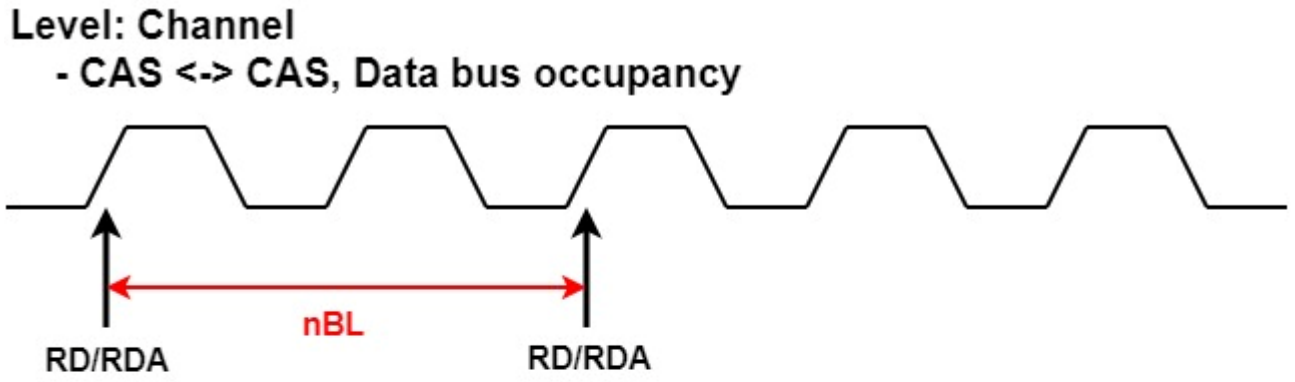
- *PHY와 Controller는 User Logic과 함께* 일반적으로 동일한 FPGA 또는 ASIC의 일부
- User Logic이 Controller에 RD 또는 WR Request를 할 때, Logical Address를 Issue
- Controller가 logical address → physical address 변환 후 PHY에게 command를 issue
- Controller는 Request를 scheduling (ex. 이미 열려있는 Row에 Access하는 Request를 우선순위로 둠)
- PHY는 Analog Driver등을 포함

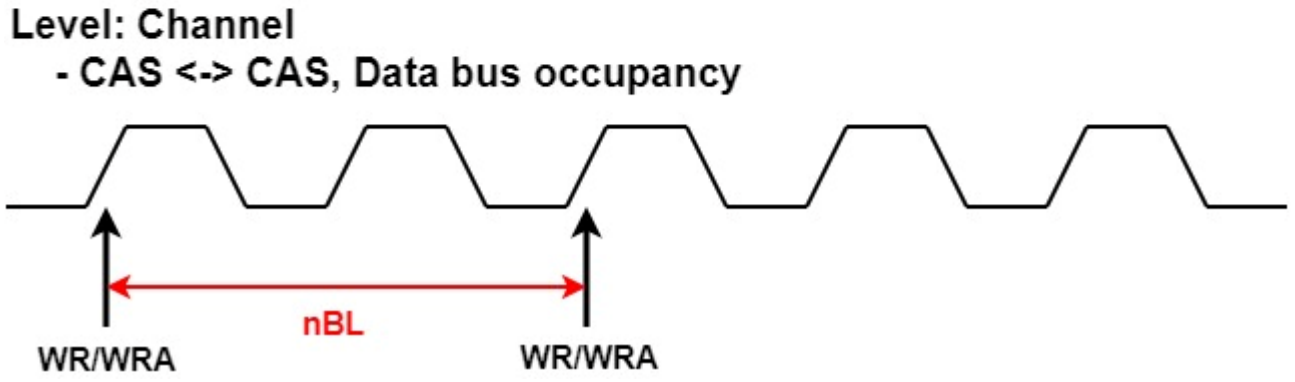


Reference: https://www.systemverilog.io/design/ddr4-basics/
Reference: BlockHammer_preventing-DRAM-rowhammer-at-low-cost_hpca21

DDR4 Summary

❑ DRAM Timings – Channel level





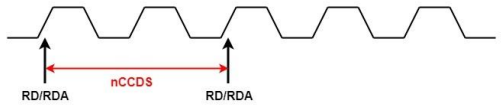
- 서로 같은 Channel에서의 Column Access
- 같은 Channel은 Data Bus 공유함

Reference: https://www.systemverilog.io/design/ddr4-basics/
Reference: BlockHammer_preventing-DRAM-rowhammer-at-low-cost_hpca21

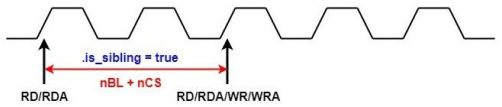
DDR4 Summary

❑ DRAM Timings – Rank level

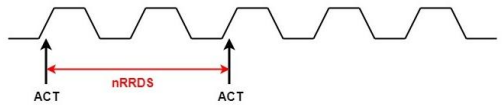
Level: Rank
- CAS <-> CAS, nCCDS is the minimal latency for column commands



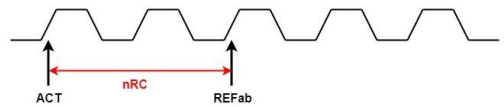
Level: Rank
- CAS <-> CAS between sibling ranks, nCS (rank switching) is needed for new DQS



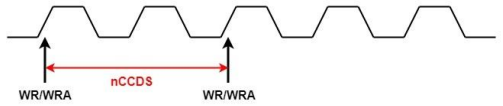
Level: Rank
- RAS <-> RAS



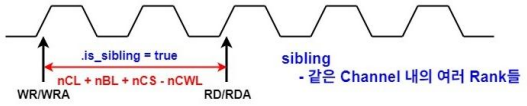
Level: Rank
- RAS <-> REF



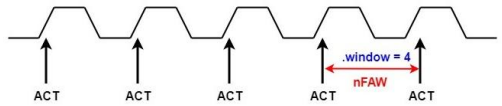
Level: Rank
- CAS <-> CAS, nCCDS is the minimal latency for column commands



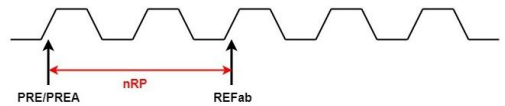
Level: Rank
- CAS <-> CAS between sibling ranks, nCS (rank switching) is needed for new DQS



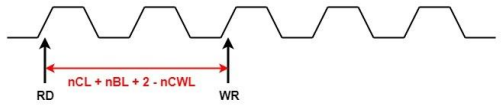
Level: Rank
- RAS <-> RAS



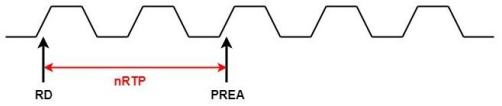
Level: Rank
- RAS <-> REF



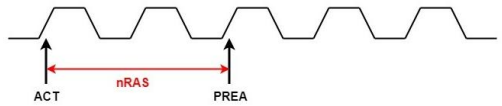
Level: Rank
- RD <-> WR, Minimum Read to Write, Assuming tWPRE = 1 tCK



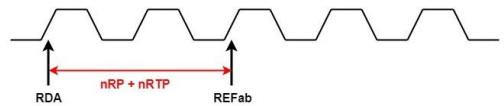
Level: Rank
- CAS <-> PREab



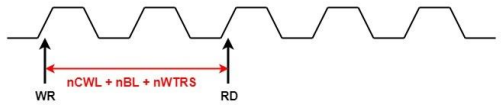
Level: Rank
- RAS <-> RAS



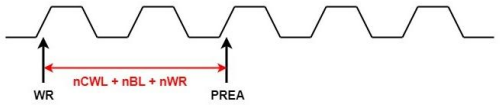
Level: Rank
- RAS <-> REF



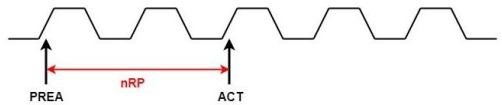
Level: Rank
- WR <-> RD, Minimum Read after Write



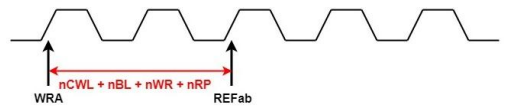
Level: Rank
- CAS <-> PREab



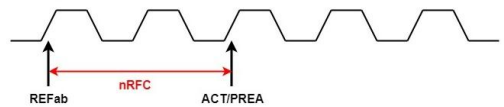
Level: Rank
- RAS <-> RAS



Level: Rank
- RAS <-> REF



Level: Rank
- RAS <-> REF

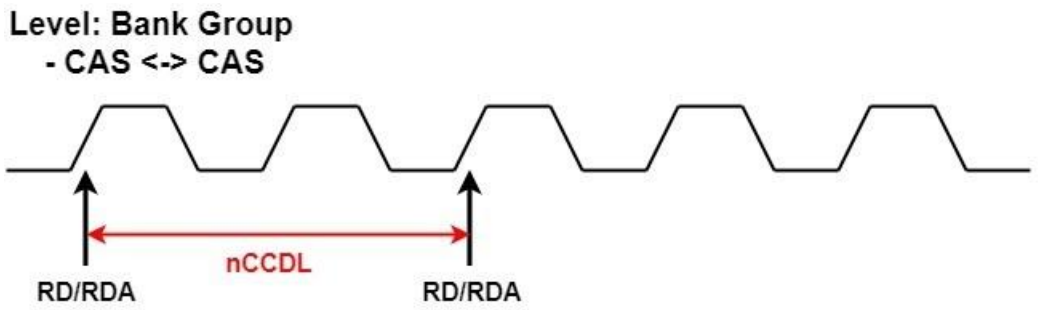


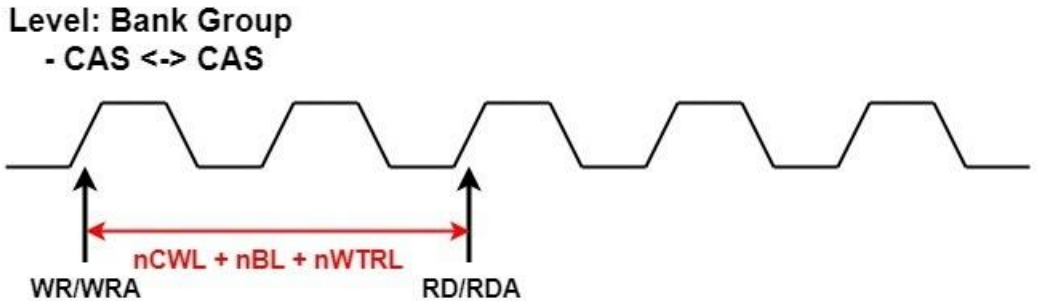
abcd

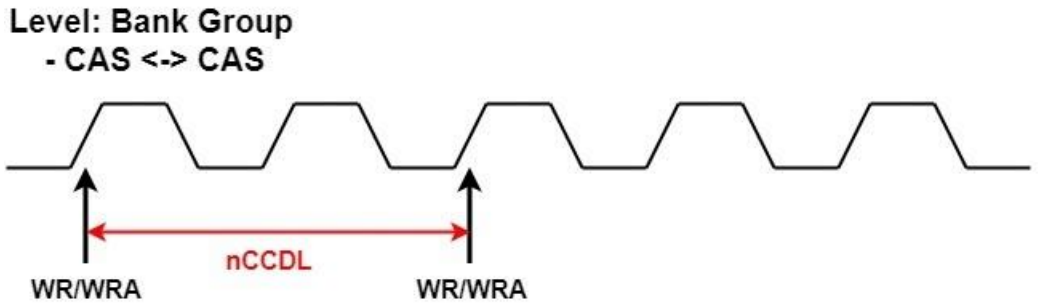
Reference: https://www.systemverilog.io/design/ddr4-basics/
Reference: BlockHammer_preventing-DRAM-rowhammer-at-low-cost_hpca21

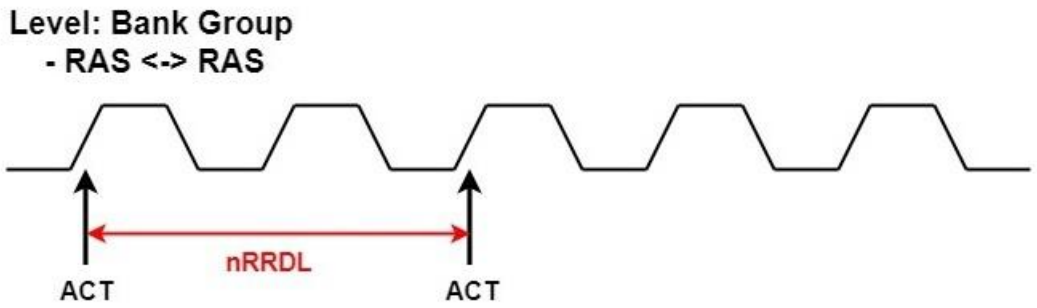
DDR4 Summary

❑ DRAM Timings – Bank Group level









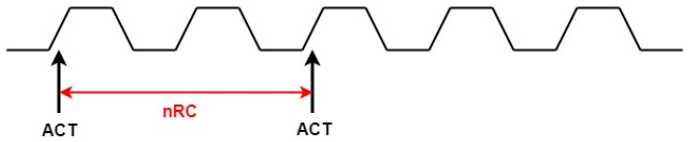
abcd

Reference: https://www.systemverilog.io/design/ddr4-basics/
Reference: BlockHammer_preventing-DRAM-rowhammer-at-low-cost_hpca21

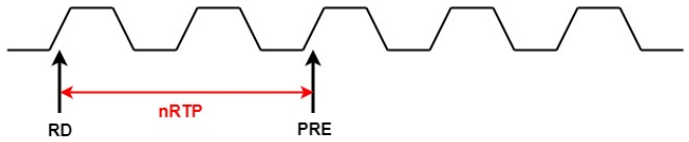
DDR4 Summary

❑ DRAM Timings – Bank level

Level: Bank

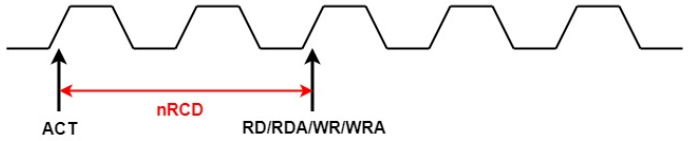


Level: Bank

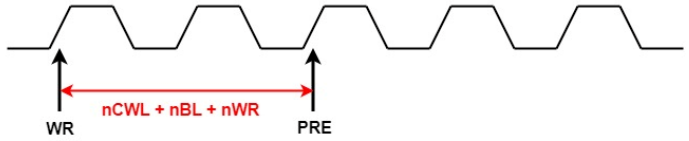


abcd

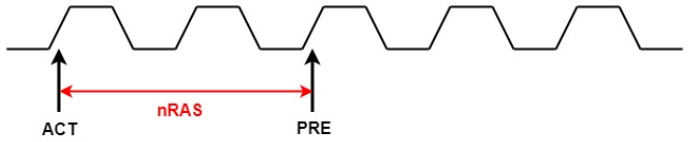
Level: Bank



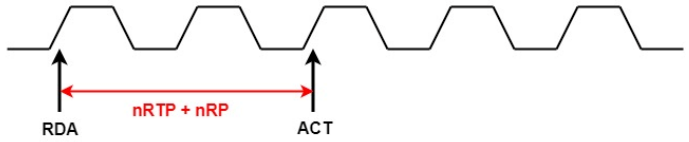
Level: Bank



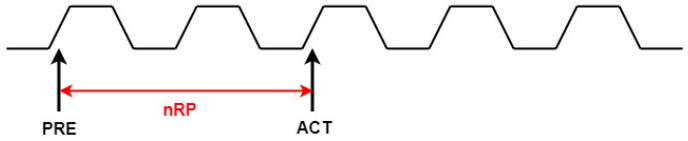
Level: Bank



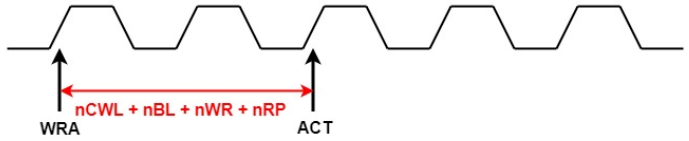
Level: Bank



Level: Bank



Level: Bank



Reference: https://www.systemverilog.io/design/ddr4-basics/
Reference: BlockHammer_preventing-DRAM-rowhammer-at-low-cost_hpca21

DDR4 Summary

❑ DRAM Timings - 겹치는 Timing Constraints

```cpp
void update_timing(int command, const AddrVec_t& addr_vec, Clk_t clk) {
  /*********************************************
   *          Update Sibling Node Timing
   *********************************************/
  if (m_node_id != addr_vec[m_level] && addr_vec[m_level] != -1) {
    for (const auto& t : m_spec->m_timing_cons[m_level][command]) {
      if (!t.sibling) {
        // not sibling timing parameter
        continue;
      }

      // update earliest schedulable time of every command
      Clk_t future = clk + t.val;
      m_cmd_ready_clk[t.cmd] = std::max(m_cmd_ready_clk[t.cmd], future);
    }
    // stop recursion
    return;
  }
```

```cpp
  /*********************************************
   *          Update Target Node Timing
   *********************************************/
  // Update history
  if (m_cmd_history[command].size()) {
    m_cmd_history[command].pop_back();
    m_cmd_history[command].push_front(clk);
  }

  for (const auto& t : m_spec->m_timing_cons[m_level][command]) {
    if (t.sibling) {
      continue;
    }

    // Get the oldest history
    Clk_t past = m_cmd_history[command][t.window-1];
    if (past < 0) {
      // not enough history
      continue;
    }

    // update earliest schedulable time of every command
    Clk_t future = past + t.val;
    m_cmd_ready_clk[t.cmd] = std::max(m_cmd_ready_clk[t.cmd], future);
  }

  if (!m_child_nodes.size()) {
    // stop recursion: updated all levels
    return;
  }

  // recursively update all of my children
  for (auto child : m_child_nodes) {
    child->update_timing(command, addr_vec, clk);
  }
};
```

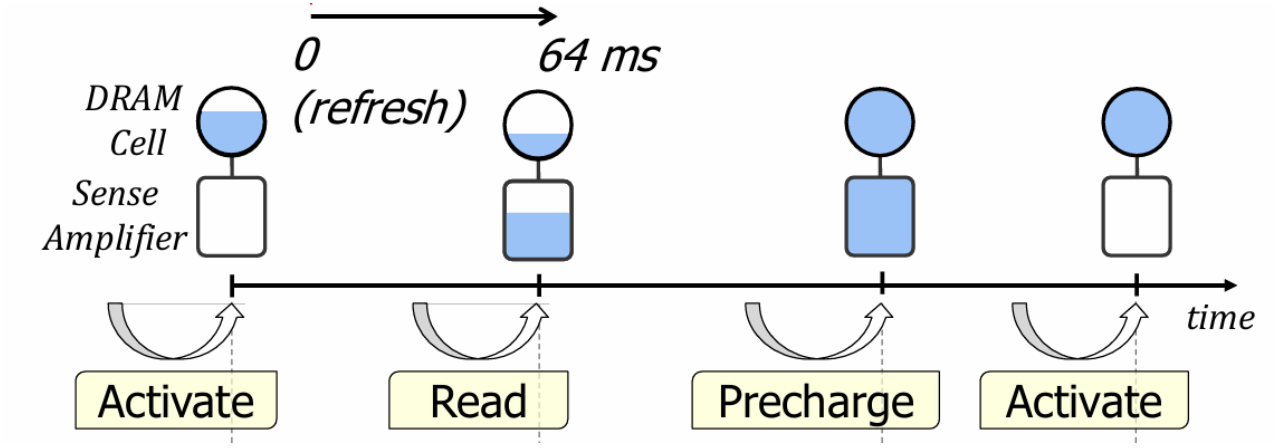
Timing Constraints 조건 겹칠 시

- **Max 값을 적용**

Reference: https://www.systemverilog.io/design/ddr4-basics/
Reference: BlockHammer_preventing-DRAM-rowhammer-at-low-cost_hpca21

DRAM Operations & States

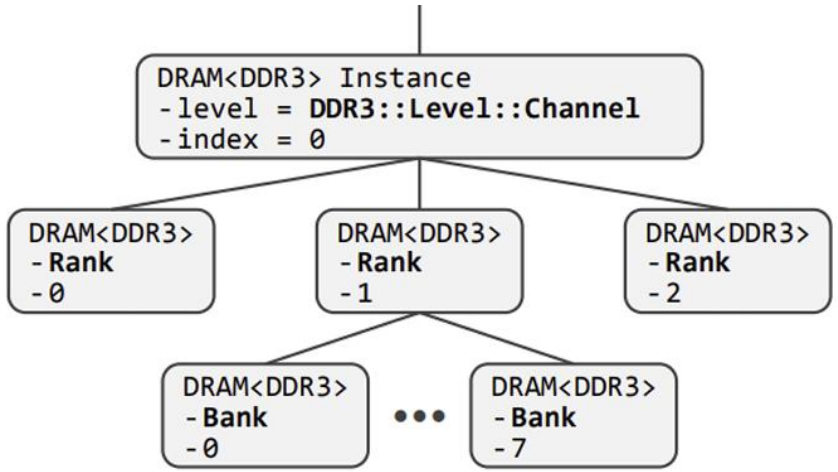
❑ DRAM Operations & States



- **Main DRAM states**
  - **Activate**
  - **Read/Write**
  - **Precharge**

```
1  // DRAM.h
2  template <typename T>
3  class DRAM {
4    DRAM<T>* parent;
5    vector<DRAM<T>*>
6      children;
7    T::Level level;
8    int index;
9    // more code...
10 };
```

```
1  // DDR3.h/cpp
2  class DDR3 {
3    enum class Level {
4      Channel, Rank, Bank,
5      Row, Column, MAX
6    };
7
8
9    // more code...
10 };
```



How to measure off-chip power?

❑ Analysis based on the paper below

BlockHammer: Preventing RowHammer at Low Cost by Blacklisting Rapidly-Accessed DRAM Rows

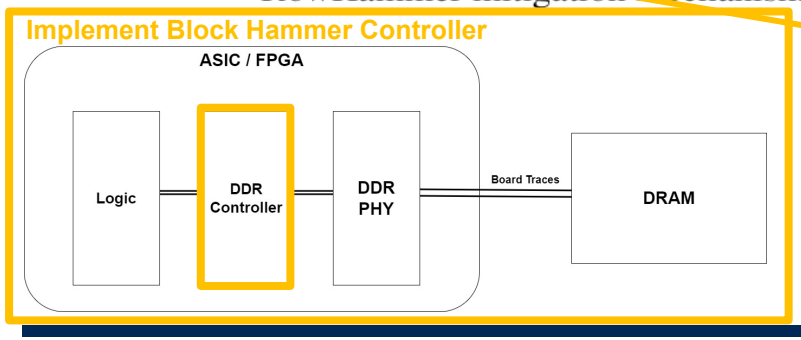
- **Section 6. Hardware Complexity Analysis**
  - **In case of on-chip analysis**

6. Hardware Complexity Analysis

We evaluate BlockHammer's (1) chip area, static power, and access energy consumption using CACTI [99] and (2) circuit latency using Synopsys DC [143]. We demonstrate that BlockHammer's physical costs are competitive with state-of-the-art RowHammer mitigation mechanisms.

6.2. Latency Analysis

We implement BlockHammer in Verilog HDL and synthesize our design using Synopsys DC [143] with a 65 nm process technology to evaluate the latency impact on memory accesses. According to our RTL model, which we open source [124], BlockHammer responds to an *"Is this ACT RowHammer-safe?"* query (❶ in Figure 2) in only 0.97 ns. This latency can be hidden because it is one-to-two orders of magnitude smaller than the row access latency (e.g., 45–50 ns) that DRAM standards (e.g., DDRx, LPDDRx, GDDRx) enforce [36, 53, 55].



Mitigation Mechanism	N_{RH}=32K* SRAM KB	CAM KB	Area mm²	Area % CPU	Access Energy (pJ)	Static Power (mW)	N_{RH}=1K SRAM KB	CAM KB	Area mm²	Area % CPU	Access Energy (pJ)	Static Power (mW)
BlockHammer	**51.48**	**1.73**	**0.14**	**0.06**	**20.30**	**22.27**	**441.33**	**55.58**	**1.57**	**0.64**	**99.64**	**220.99**
Dual counting Bloom filters	48.00	-	0.11	0.04	18.11	19.81	384.00	-	0.74	0.30	86.29	158.46
H3 hash functions	-	-	< 0.01	< 0.01	-	-	-	-	< 0.01	< 0.01	-	-
Row activation history buffer	1.73	1.73	0.03	0.01	1.83	2.05	55.58	55.58	0.83	0.34	12.99	62.12
AttackThrottler counters	1.75	-	< 0.01	< 0.01	0.36	0.41	1.75	-	< 0.01	< 0.01	0.36	0.41
PARA [73]	**-**	**-**	**< 0.01**	**-**	**-**	**-**	**-**	**-**	**< 0.01**	**-**	**-**	**-**
ProHIT [137]*	**-**	**0.22**	**< 0.01**	**<0.01**	**3.67**	**0.14**	×	×	×	×	×	×
MrLoc [161]*	**-**	**0.47**	**< 0.01**	**<0.01**	**4.44**	**0.21**	×	×	×	×	×	×
CBT [132]	**16.00**	**8.50**	**0.20**	**0.08**	**9.13**	**35.55**	**512.00**	**272.00**	**3.95**	**1.60**	**127.93**	**535.50**
TWiCE [84]	**23.10**	**14.02**	**0.15**	**0.06**	**7.99**	**21.28**	**738.32**	**448.27**	**5.17**	**2.10**	**124.79**	**631.98**
Graphene [113]	**-**	**5.22**	**0.04**	**0.02**	**40.67**	**3.11**	**-**	**166.03**	**1.14**	**0.46**	**917.55**	**93.96**

* PRoHIT [137] and MRLoc [161] do *not* provide a concrete discussion on how to adjust their empirically-determined parameters for different N_{RH} values. Therefore, we (1) report their values for a fixed design point that each paper provides for N_{RH}=2K and (2) mark values we cannot estimate using an ×.

Table 4: Per-rank area, access energy, and static power of BlockHammer vs. state-of-the-art RowHammer mitigation mechanisms.

How to measure off-chip power?

❑ Analysis based on the paper below

BlockHammer: Preventing RowHammer at Low Cost by Blacklisting Rapidly-Accessed DRAM Rows

- **Section 6. Hardware Complexity Analysis**
  - **In case of off-chip analysis**

7. Experimental Methodology

We evaluate BlockHammer's effect on a typical DDR4-based memory subsystem's performance and energy consumption as compared to six prior RowHammer mitigation mechanisms [73, 84, 113, 132, 137, 161]. We use Ramulator [77, 125] for performance evaluation and DRAMPower [18] to estimate DRAM energy consumption. We open-source our infrastructure, which implements both BlockHammer and six state-of-the-art RowHammer mitigation mechanisms [124]. Table 5 shows our system configuration.

Processor	3.2 GHz, {1,8} core, 4-wide issue, 128-entry instr. window
Last-Level Cache	64-byte cache line, 8-way set-associative, 16 MB
Memory Controller	64-entry each read and write request queues; Scheduling policy: FR-FCFS [122, 164]; Address mapping: MOP [60]
Main Memory	DDR4, 1 channel, 1 rank, 4 bank groups, 4 banks/bank group, 64K rows/bank

Table 5: Simulated system configuration.

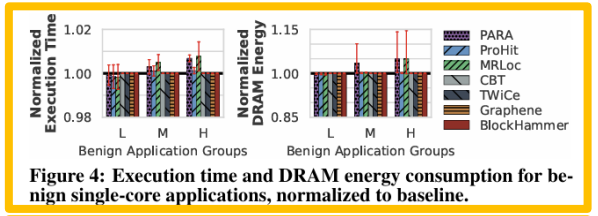


Figure 4: Execution time and DRAM energy consumption for benign single-core applications, normalized to baseline.

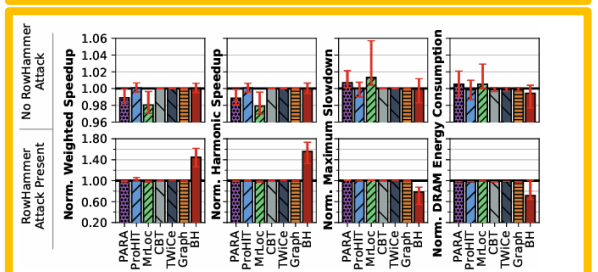
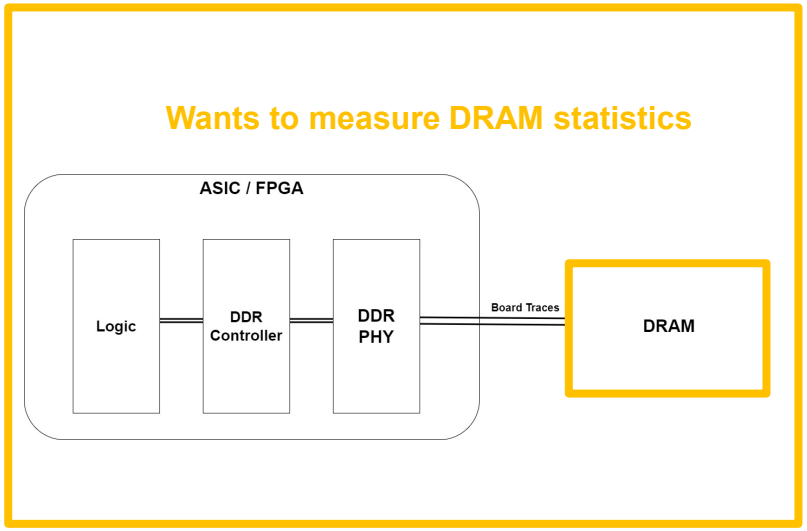
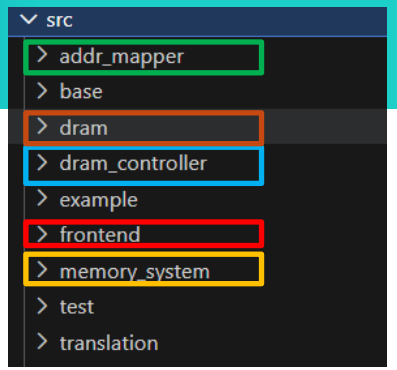


Figure 5: Performance and DRAM energy consumption for multiprogrammed workloads, normalized to baseline.



src files <=> DRAM Operation



❑ Simulation Configuration

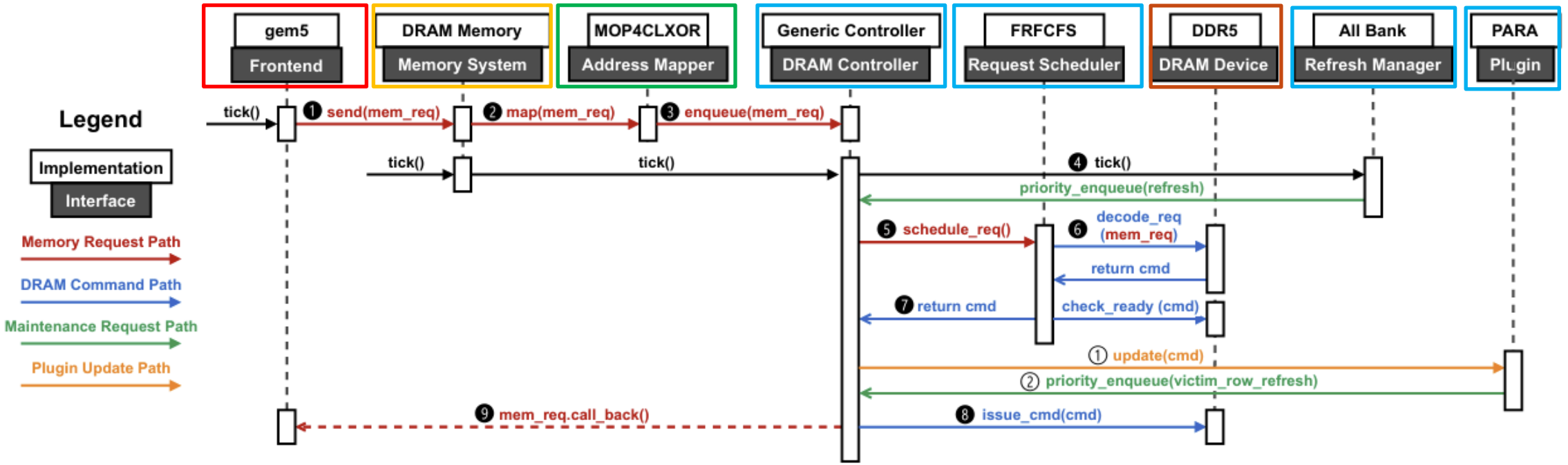


Fig. 1: High-level software architecture of Ramulator 2.0 using an example DDR5 system configuration

src files <=> DRAM Operation

❑ Simulation Configuration

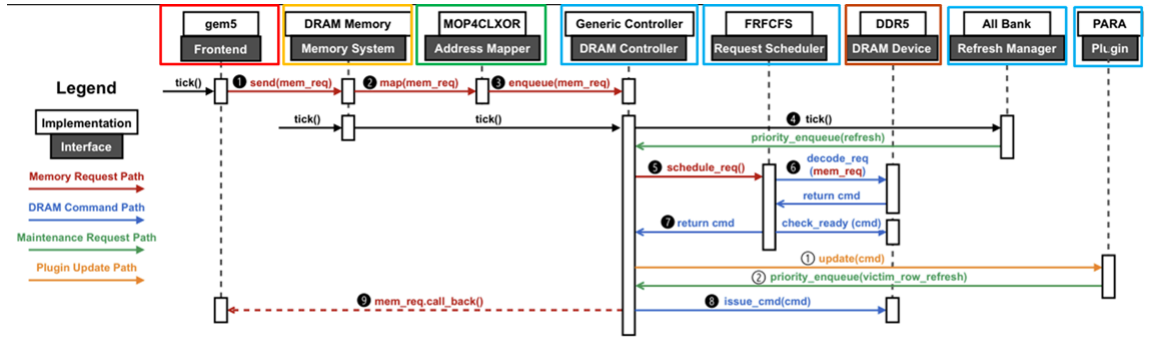
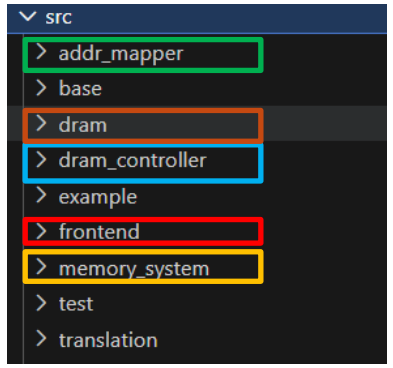


Fig. 1: High-level software architecture of Ramulator 2.0 using an example DDR5 system configuration



1. **Requests are sent**: Front-end(trace file)에서 Memory Request를 보냄
2. **Memory Addresses are Mapped**: Address Mapper가 Request Address를 DRAM 구조에 맞게 변환
3. **Enqueue**: DRAM Ctrlr의 Buffer에 Request를 넣음
4. **DRAM Ctrlr -** Ticking Refresh Manager: Ctrlr가 Refresh Manager를 호출해 high-priority maintenance request(ex. Refresh)을 추가
5. **DRAM Ctrlr - Request Scheduling**: Request Scheduler에게 최적의 Request을 선택하라고 요청
6. **DRAM Device가 Request 확인**: Scheduler가 DRAM Device Model을 참조해 적합한 Command를 Decode
7. **Issue Command**: DRAM Ctrlr가 DRAM Command를 보냄
8. **Updates the behavior and timing information:** DRAM Command Issue시 State & Timing이 Update
9. **Notify the frontend**: Memory Request가 끝나면 callback으로 frontend에 알림

main function

❑ main.cpp

```cpp
int main(int argc, char* argv[]) {
  // Parse command line arguments
  argparse::ArgumentParser program("Ramulator", "2.0");
  program.add_argument("-c", "--config").metavar("\"dumped YAML configuration\"")
    .help("String dump of the yaml configuration.");
  program.add_argument("-f", "--config_file").metavar("path-to-configuration-file")
    .help("Path to a YAML configuration file.");
  program.add_argument("-p", "--param").metavar("KEY=VALUE")
    .append()
    .help("Specify parameter to override in the configuration file. Repeat this option to change multiple parameters.");
```

•
•
•

```cpp
  // Connect the frontend and the memory system together,
  // this recursively calls the "setup" function in all instaniated components
  // so that they can get each other's parameters (if needed) after their initialization
  frontend->connect_memory_system(memory_system);
  memory_system->connect_frontend(frontend);

  // Get the relative clock ratio between the frontend and memory system
  int frontend_tick = frontend->get_clock_ratio();
  int mem_tick = memory_system->get_clock_ratio();

  int tick_mult = frontend_tick * mem_tick;

  for (uint64_t i = 0;; i++) {
    if (((i % tick_mult) % mem_tick) == 0) {
      frontend->tick();
    }

    if (frontend->is_finished()) {
      break;
    }

    if ((i % tick_mult) % frontend_tick == 0) {
      memory_system->tick();
    }
  }

  // Finalize the simulation. Recursively print all statistics from all components
  frontend->finalize();
  memory_system->finalize();

  return 0;
}
```

main.cpp

1. **Argument 받는 부분**
   - **Options**
   1. -c: command line dump
   2. -f: YAML document
   3. -p: overriding parameters in a YAML document

2. **Long for loop를 통한 tick() 기반 simul**
   1. frontend(core)가 발행한 예상 instructions들을 모두 처리시 is_finished()가 true가 됨

yaml file

❑example_config.yaml

```yaml
Frontend:
  impl: SimpleO3
  clock_ratio: 8
  num_expected_insts: 500000
  traces:
    - example_inst.trace

  Translation:
    impl: RandomTranslation
    max_addr: 2147483648

MemorySystem:
  impl: GenericDRAM
  clock_ratio: 3

  DRAM:
    impl: DDR4
    org:
      preset: DDR4_8Gb_x8
      channel: 1
      rank: 2
    timing:
      preset: DDR4_2400R

  Controller:
    impl: Generic
    Scheduler:
      impl: FRFCFS
    RefreshManager:
      impl: AllBank
    RowPolicy:
      impl: ClosedRowPolicy
      cap: 4
    plugins:

  AddrMapper:
    impl: RoBaRaCoCh
```

1. Frontend Interface(IFrontEnd) 부분

- trace file에서 Instruction읽고, Memory Request 생성
- *impl: SimpleO3*
  ⇒ Simple Out-of-Order (O3) CPU
- *clock_ratio: 8*
  ⇒ global CLK 대비 Frontend CLK 속도
- *num_expected_insts: 500000*
  ⇒ Simulation이 해당 instruction 수에 도달 시 종료
- *traces*
  ⇒ Instruction trace file(include memory access Inst)
- *impl: RandomTranslation*
  ⇒ Physical Memory ↔ Virtual Memory 변환
  ⇒ System의 Page Table 등을 간단히 Modeling
- *max_addr: 2147483648*
  ⇒ Translation 시 address overflow 방지

yaml file

❑example_config.yaml

```yaml
Frontend:
  impl: SimpleO3
  clock_ratio: 8
  num_expected_insts: 500000
  traces:
    - example_inst.trace

Translation:
  impl: RandomTranslation
  max_addr: 2147483648

MemorySystem:
  impl: GenericDRAM
  clock_ratio: 3

  DRAM:
    impl: DDR4
    org:
      preset: DDR4_8Gb_x8
      channel: 1
      rank: 2
    timing:
      preset: DDR4_2400R

  Controller:
    impl: Generic
    Scheduler:
      impl: FRFCFS
    RefreshManager:
      impl: AllBank
    RowPolicy:
      impl: ClosedRowPolicy
      cap: 4
    plugins:

  AddrMapper:
    impl: RoBaRaCoCh
```

2. MemorySystem Interface 부분

- Frontend의 Request를 받아 DRAM Ctrlr을 통해 처리
- Latency, en/dequeue, Timing Constraints 처리

- ***impl: GenericDRAM***
  ⇒ 기본 DRAM 기반 System, Ctrlr와 DRAM을 통합

- ***clock_ratio: 3***
  ⇒ global CLK 대비 MemorySystem CLK 속도
  ⇒ 현재: DRAM이 CPU보다 느린 System (= 3:8)

yaml file

❑example_config.yaml

```yaml
Frontend:
  impl: SimpleO3
  clock_ratio: 8
  num_expected_insts: 500000
  traces:
    - example_inst.trace

Translation:
  impl: RandomTranslation
  max_addr: 2147483648

MemorySystem:
  impl: GenericDRAM
  clock_ratio: 3

  DRAM:
    impl: DDR4
    org:
      preset: DDR4_8Gb_x8
      channel: 1
      rank: 2
    timing:
      preset: DDR4_2400R

  Controller:
    impl: Generic
    Scheduler:
      impl: FRFCFS
    RefreshManager:
      impl: AllBank
    RowPolicy:
      impl: ClosedRowPolicy
      cap: 4
    plugins:

  AddrMapper:
    impl: RoBaRaCoCh
```

2. MemorySystem Interface 부분

- DRAM Section
- *impl: DDR4*
  ⇒ tick() 시 Timing Check / Command Issue 실행.
- *org: DDR4_8Gb_x8*
  ⇒ 현재 DRAM preset - 8Gb 용량, x8bit data bus
  ⇒ Channel/Rank 설정 시 기본 Preset설정을 Override 함
- *timing: DDR4_2400R*
  ⇒ Timing preset - nRCD등의 Timing Contraint 정의
  ⇒ 이를 이용해 tick() 시 Latency 계산

```
inline static const std::map<std::string, std::vector<int>> timing_presets = {
  //   name         rate   nBL  nCL  nRCD  nRP   nRAS  nRC   nWR  nRTP nCWL nCCDS nCCDL nRRDS nRRDL nWTRS nW
  {"DDR4_1600J",  {1600,   4,  10,  10,   10,   28,   38,   12,   6,   9,    4,    5,   -1,   -1,   2,
  {"DDR4_1600K",  {1600,   4,  11,  11,   11,   28,   39,   12,   6,   9,    4,    5,   -1,   -1,   2,
  {"DDR4_1600L",  {1600,   4,  12,  12,   12,   28,   40,   12,   6,   9,    4,    5,   -1,   -1,   2,
  {"DDR4_1866L",  {1866,   4,  12,  12,   12,   32,   44,   14,   7,   10,   4,    5,   -1,   -1,   3,
  {"DDR4_1866M",  {1866,   4,  13,  13,   13,   32,   45,   14,   7,   10,   4,    5,   -1,   -1,   3,
  {"DDR4_1866N",  {1866,   4,  14,  14,   14,   32,   46,   14,   7,   10,   4,    5,   -1,   -1,   3,
  {"DDR4_2133N",  {2133,   4,  14,  14,   14,   36,   50,   16,   8,   11,   4,    6,   -1,   -1,   3,
  {"DDR4_2133P",  {2133,   4,  15,  15,   15,   36,   51,   16,   8,   11,   4,    6,   -1,   -1,   3,
  {"DDR4_2133R",  {2133,   4,  16,  16,   16,   36,   52,   16,   8,   11,   4,    6,   -1,   -1,   3,
  {"DDR4_2400P",  {2400,   4,  15,  15,   15,   39,   54,   18,   9,   12,   4,    6,   -1,   -1,   3,
  {"DDR4_2400R",  {2400,   4,  16,  16,   16,   39,   55,   18,   9,   12,   4,    6,   -1,   -1,   3,
  {"DDR4_2400U",  {2400,   4,  17,  17,   17,   39,   56,   18,   9,   12,   4,    6,   -1,   -1,   3,
  {"DDR4_2400T",  {2400,   4,  18,  18,   18,   39,   57,   18,   9,   12,   4,    6,   -1,   -1,   3,
```

```
class DDR4 : public IDRAM, public Implementation {
  RAMULATOR_REGISTER_IMPLEMENTATION(IDRAM, DDR4, "DDR4", "DDR4 Device Model")

public:
  inline static const std::map<std::string, Organization> org_presets = {
    //   name         density   DQ   Ch Ra Bg Ba   Ro     Co
    {"DDR4_2Gb_x4",   {2<<10,   4,  {1, 1, 4, 4, 1<<15, 1<<10}}},
    {"DDR4_2Gb_x8",   {2<<10,   8,  {1, 1, 4, 4, 1<<14, 1<<10}}},
    {"DDR4_2Gb_x16",  {2<<10,   16, {1, 1, 2, 4, 1<<14, 1<<10}}},
    {"DDR4_4Gb_x4",   {4<<10,   4,  {1, 1, 4, 4, 1<<16, 1<<10}}},
    {"DDR4_4Gb_x8",   {4<<10,   8,  {1, 1, 4, 4, 1<<15, 1<<10}}},
    {"DDR4_4Gb_x16",  {4<<10,   16, {1, 1, 2, 4, 1<<15, 1<<10}}},
    {"DDR4_8Gb_x4",   {8<<10,   4,  {1, 1, 4, 4, 1<<17, 1<<10}}},
    {"DDR4_8Gb_x8",   {8<<10,   8,  {1, 1, 4, 4, 1<<16, 1<<10}}},
    {"DDR4_8Gb_x16",  {8<<10,   16, {1, 1, 2, 4, 1<<16, 1<<10}}},
    {"DDR4_16Gb_x4",  {16<<10,  4,  {1, 1, 4, 4, 1<<18, 1<<10}}},
    {"DDR4_16Gb_x8",  {16<<10,  8,  {1, 1, 4, 4, 1<<17, 1<<10}}},
    {"DDR4_16Gb_x16", {16<<10,  16, {1, 1, 2, 4, 1<<17, 1<<10}}},
  };
```

```
class DDR4 : public IDRAM, public Implementation {
    void tick() override {

  void init() override {
    RAMULATOR_DECLARE_SPECS();
    set_organization();
    set_timing_vals();

    set_actions();
    set_preqs();
    set_rowhits();
    set_rowopens();
    set_powers();

    create_nodes();
  };
```

yaml file

❑example_config.yaml

```yaml
Frontend:
  impl: SimpleO3
  clock_ratio: 8
  num_expected_insts: 500000
  traces:
    - example_inst.trace

  Translation:
    impl: RandomTranslation
    max_addr: 2147483648

MemorySystem:
  impl: GenericDRAM
  clock_ratio: 3

  DRAM:
    impl: DDR4
    org:
      preset: DDR4_8Gb_x8
      channel: 1
      rank: 2
    timing:
      preset: DDR4_2400R

  Controller:
    impl: Generic
    Scheduler:
      impl: FRFCFS
    RefreshManager:
      impl: AllBank
    RowPolicy:
      impl: ClosedRowPolicy
      cap: 4
    plugins:

  AddrMapper:
    impl: RoBaRaCoCh
```

2. MemorySystem Interface 부분

- **Controller Section**
- *impl: Generic*
  ⇒ Generic - 기본 Ctrlr
  ⇒ Request Queue/Scheduling 등 관리
- *Scheduler - impl: FRFCFS*
  ⇒ FRFCFS - First-Ready First-Come-First-Serve
  ⇒ 준비된 Request를 Queue에서 꺼내 우선 처리
- *RefreshManager - impl: AllBank*
  ⇒ AllBank - 모든 Bank simultaneous Refresh
- *RowPolicy - impl: ClosedRowPolicy*
  ⇒ ClosedRowPolicy - 사용 후 Row 즉시 닫음(Precharge)
  ⇒ cap:4 - 열려있는 Row 최대 수 제한
- *plugins*
  ⇒ 현재 Ramulator에서는 Row Hammering 완화 기법을 plugin으로 제공해줌

yaml file

❑example_config.yaml

```yaml
Frontend:
  impl: SimpleO3
  clock_ratio: 8
  num_expected_insts: 500000
  traces:
    - example_inst.trace

Translation:
  impl: RandomTranslation
  max_addr: 2147483648

MemorySystem:
  impl: GenericDRAM
  clock_ratio: 3

  DRAM:
    impl: DDR4
    org:
      preset: DDR4_8Gb_x8
      channel: 1
      rank: 2
    timing:
      preset: DDR4_2400R

  Controller:
    impl: Generic
    Scheduler:
      impl: FRFCFS
    RefreshManager:
      impl: AllBank
    RowPolicy:
      impl: ClosedRowPolicy
      cap: 4
    plugins:

  AddrMapper:
    impl: RoBaRaCoCh
```

2. MemorySystem Interface 부분

- AddrMapper Section

- ***impl: RoBaRaCoCh***
  - ⇒ Row-Bank-Rank-Column-Channel Mapping Scheme
  - ⇒ Requested Addres 변환(Physical → DRAM Vector)

  [Physical → DRAM Vector Example]
  - ⇒ Physical Address: 0x12345678
  - ⇒ DRAM Vector:
    [Channel:0, Rank:1, Bank:2, Row:128, Column:512]

trace file

❑ example_inst.trace & trace.cpp & core.cpp

example_inst.trace

1	3	20734016	
2	1	20846400	
3	6	20734208	
4	1	20846400	
5	8	20841280	20841280
6	0	20734144	
7	2	20918976	20734016
8	1	20846400	
9			

- simpleO3 CPU model기준
  - Ramulator는 “Memory” Simulator
  - Memory 명령어만 취급하기에, 3가지로만 Instruction을 분리한다.
    1. Not Memory Operation
    2. Load
    3. Store
- 각 line은 load(& store)동작을 나타낸다.
- 1st line : 3cycle동안 stall → load
- 5th line : 8cycle동안 stall → load → store
- 따라서, simpleO3 기반 trace파일:
  - 1st column은 Not Memory Operation Cycle 수 (or ticks 수)
  - 2nd column은 load operation address
  - 3rd column은 store operation address

trace file

❑ example_inst.trace & trace.cpp & core.cpp

- Trace file을 arg로 받아, Frontend (ex: simpleO3.cpp)에서 처리되어 메모리 접근 request를 생성
- simpleO3 CPU model기준: trace의 token[0]: bubble / token[1]: load address / token[2]: store address

trace file result

❑ ./ramulator2 -f ./example_config.yaml

```
root@947e591ed45c:/workspace# ./ramulator2 -f ./example_config.yaml
Frontend:
  impl: SimpleO3
  memory_access_cycles_recorded_core_0: 61
  cycles_recorded_core_0: 216815
  llc_mshr_unavailable: 0
  llc_read_misses: 37
  llc_read_access: 133336
  llc_write_misses: 8
  llc_write_access: 33334
  llc_eviction: 0
  num_expected_insts: 500000
  Translation:
    impl: RandomTranslation

MemorySystem:
  impl: GenericDRAM
  total_num_other_requests: 0
  total_num_write_requests: 0
  total_num_read_requests: 6
  memory_system_cycles: 81306
  DRAM:
    impl: DDR4
  AddrMapper:
    impl: RoBaRaCoCh

Controller:
  impl: Generic
  id: Channel 0
  avg_read_latency_0: 46.5
  read_queue_len_avg_0: 0.00232455181
  write_queue_len_0: 0
  queue_len_0: 245
  num_other_reqs_0: 0
  num_write_reqs_0: 0
  read_latency_0: 279
  priority_queue_len_avg_0: 0.000688756059
  row_hits_0: 2
  priority_queue_len_0: 56
  row_misses_0: 4
  row_conflicts_0: 0
  read_row_misses_0: 4
  queue_len_avg_0: 0.00301330769
  read_row_conflicts_core_0: 0
  read_row_hits_0: 2
  write_queue_len_avg_0: 0
  read_row_conflicts_0: 0
  write_row_misses_0: 0
  write_row_conflicts_0: 0
  read_queue_len_0: 189
  write_row_hits_0: 0
  read_row_hits_core_0: 2
  read_row_misses_core_0: 4
  num_read_reqs_0: 6
  Scheduler:
    impl: FRFCFS
  RefreshManager:
    impl: AllBank

RowPolicy:
  impl: ClosedRowPolicy
  num_close_reqs: 0
```

- 기본 yaml file setting을 이용.
  => trace file의 Inst 실행결과가 분석됨

- 1. Frontend:
  CPU ↔ Memory<Cache> 접근 분석

- 2. MemorySystem<DRAM> 관점 분석
  - Request 수
  - Cycle 수
  - Latency
  - Queue 길이
  - Row Hit / Miss

trace file result analysis

example_inst.trace

1	3	20734016	
2	1	20846400	
3	6	20734208	
4	1	20846400	
5	8	20841280	20841280
6	0	20734144	
7	2	20918976	20734016
8	1	20846400	
9			

❑ ./ramulator2 -f ./example_config.yaml

- 분석 시 사용한 trace file은 오른쪽의 example_inst이다.
- 예상 simulation결과는 다음과 같아야 한다 → *이를 실제 Results와 비교한다.*
- 8번의 load (address: 0x20846400가 3번 나온다)
- 2번의 write (1st Inst의 address, 5th Inst의 address에 Write [이미 load된 address 접근 → cache hit])
  - DRAM 관점

    → 이를 통해서 6번의 DRAM Read Access를 요구하는 것을 예측 가능

    - 나머지 중복 load들은 cache line에 hit되어 DRAM Access X

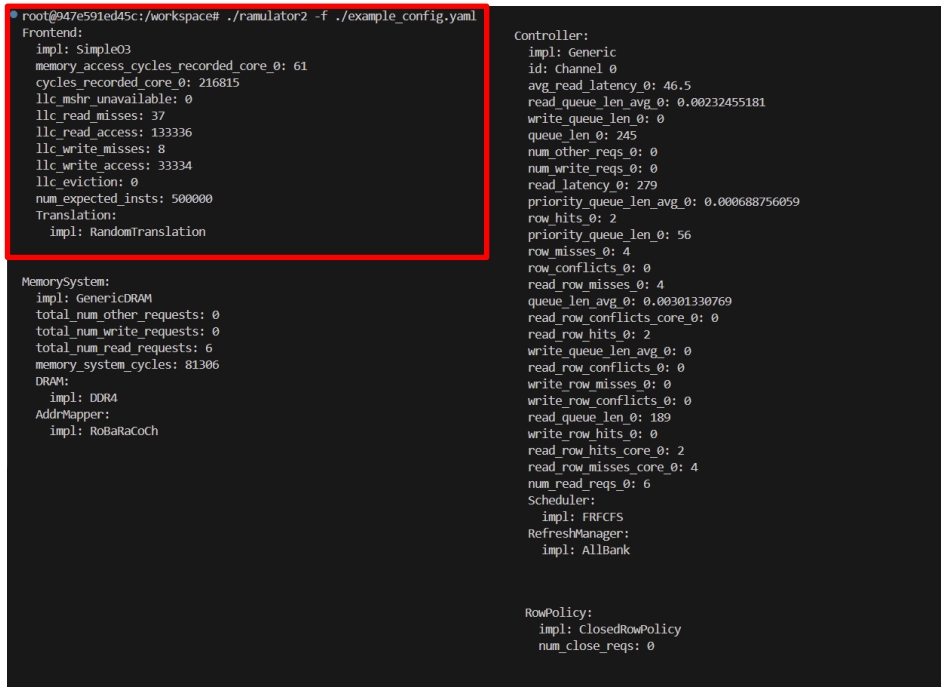
    → Write의 경우 이미 Load된 address에 write하므로 cache이용 : 0번의 DRAM Write Access
  - SRAM(cache) 관점

    → 적은 memory 주소만 반복 접근 하므로 cache eviction X (이전에 load된 addr들은 다 hit 됨)

    → Initial Read Misses: 6번 / Initial Write Misses: 0번

trace file result analysis

❑ ./ramulator2 -f ./example_config.yaml




```
52  const SimpleO3Core::Trace::Inst& SimpleO3Core::Trace::get_next_inst(
53      const Inst& inst = m_trace[m_curr_trace_idx];
54      m_curr_trace_idx = (m_curr_trace_idx + 1) % m_trace_length;
55      return inst;
56  }
```

짧은 trace지만 반복 실행 되기에 많은 access 발생

1. **Frontend**
   - Last Level Cache(LLC) 분석
     - Core 0의 Memory(Cache 포함) 접근 Cycles
     - LLC의 Read Request Cache Miss 수
     - LLC의 Read Access 수
     - Miss Status Handling Register(MSHR) unavailable cnt
       → MSHR가 가득 차서 Request 거부된 횟수
     - LLC의 Write Request Cache Miss 수
     - LLC의 Write Access 수
     - LLC의 Eviction 수
       → Cache Line 교체된 수 (DRAM Write 유발)

이전의 예상치와 다르게 LLC의 miss수가 훨씬 많음
→ dependency check, reorder buffer 등 때문일 수 있음
→ LLC operation 코드 분석이 필요 → OoO CPU의 특성 때문임
→ bool SimpleO3LLC::send(Request req)

trace file result analysis

❑ ./ramulator2 -f ./example_config.yaml

```
root@947e591ed45c:/workspace# ./ramulator2 -f ./example_config.yaml
Frontend:
  impl: SimpleO3
  memory_access_cycles_recorded_core_0: 61
  cycles_recorded_core_0: 216815
  llc_mshr_unavailable: 0
  llc_read_misses: 37
  llc_read_access: 133336
  llc_write_misses: 8
  llc_write_access: 33334
  llc_eviction: 0
  num_expected_insts: 500000
  Translation:
    impl: RandomTranslation

MemorySystem:
  impl: GenericDRAM
  total_num_other_requests: 0
  total_num_write_requests: 0
  total_num_read_requests: 6
  memory_system_cycles: 81306
  DRAM:
    impl: DDR4
  AddrMapper:
    impl: RoBaRaCoCh

Controller:
  impl: Generic
  id: Channel 0
  avg_read_latency_0: 46.5
  read_queue_len_avg_0: 0.00232455181
  write_queue_len_0: 0
  queue_len_0: 245
  num_other_reqs_0: 0
  num_write_reqs_0: 0
  read_latency_0: 279
  priority_queue_len_avg_0: 0.000688756059
  row_hits_0: 2
  priority_queue_len_0: 56
  row_misses_0: 4
  row_conflicts_0: 0
  read_row_misses_0: 4
  queue_len_avg_0: 0.00301330769
  read_row_conflicts_core_0: 0
  read_row_hits_0: 2
  write_queue_len_avg_0: 0
  read_row_conflicts_0: 0
  write_row_misses_0: 0
  write_row_conflicts_0: 0
  read_queue_len_0: 189
  write_row_hits_0: 0
  read_row_hits_core_0: 2
  read_row_misses_core_0: 4
  num_read_reqs_0: 6
  Scheduler:
    impl: FRFCFS
  RefreshManager:
    impl: AllBank

RowPolicy:
  impl: ClosedRowPolicy
  num_close_reqs: 0
```

2. MemorySystem Interface

- **DRAM 분석**
  - **Read/Write 외 other requests 수** *이전의 예상치와 일치*
  - **write requests 수 → 현재 0 (cache eviction 없었기에)**
  - **read requests 수 → 현재 6 (initial cache line load)**
  - **Memory system cycles**
    → Memory CLK과 Proc CLK이 다르기에 (= 3:8)

DRAM CLK : Processor CLK = 3:8


```
108     inline static constexpr ImplDef m_requests = {
109       "read", "write", "all-bank-refresh", "open-row", "close-row"
110     };
111
112     inline static const ImplLUT m_request_translations = LUT (
113       m_requests, m_commands, {
114         {"read", "RD"}, {"write", "WR"}, {"all-bank-refresh", "REFab"},
115         {"open-row", "ACT"}, {"close-row", "PRE"}
116       }
117     );
```

Other requests: read/write를 제외한 open-row와 같은 request등을 의미

trace file result analysis

❑ ./ramulator2 -f ./example_config.yaml

```
root@947e591ed45c:/workspace# ./ramulator2 -f ./example_config.yaml
Frontend:
  impl: SimpleO3
  memory_access_cycles_recorded_core_0: 61
  cycles_recorded_core_0: 216815
  llc_mshr_unavailable: 0
  llc_read_misses: 37
  llc_read_access: 133336
  llc_write_misses: 8
  llc_write_access: 33334
  llc_eviction: 0
  num_expected_insts: 500000
  Translation:
    impl: RandomTranslation

MemorySystem:
  impl: GenericDRAM
  total_num_other_requests: 0
  total_num_write_requests: 0
  total_num_read_requests: 6
  memory_system_cycles: 81306
  DRAM:
    impl: DDR4
  AddrMapper:
    impl: RoBaRaCoCh
```

```
Controller:
  impl: Generic
  id: Channel 0
  avg_read_latency_0: 46.5
  read_queue_len_avg_0: 0.00232455181
  write_queue_len_0: 0
  queue_len_0: 245
  num_other_reqs_0: 0
  num_write_reqs_0: 0
  read_latency_0: 279
  priority_queue_len_avg_0: 0.000688756059
  row_hits_0: 2
  priority_queue_len_0: 56
  row_misses_0: 4
  row_conflicts_0: 0
  read_row_misses_0: 4
  queue_len_avg_0: 0.00301330769
  read_row_conflicts_core_0: 0
  read_row_hits_0: 2
  write_queue_len_avg_0: 0
  read_row_conflicts_0: 0
  write_row_misses_0: 0
  write_row_conflicts_0: 0
  read_queue_len_0: 189
  write_row_hits_0: 0
  read_row_hits_core_0: 2
  read_row_misses_core_0: 4
  num_read_reqs_0: 6
  Scheduler:
    impl: FRFCFS
  RefreshManager:
    impl: AllBank


  RowPolicy:
    impl: ClosedRowPolicy
    num_close_reqs: 0
```

2. MemorySystem Interface
   - DRAM Controller 분석
     - abcd

trace file result

❑power는 어떻게 측정하느냐면

- yaml file에 아래 drampower_enable 옵션을 넣음
- dram.h -> DDR4.cpp 의 power함수 enable됨

```yaml
DRAM:
  impl: DDR4
  org:
    preset: DDR4_8Gb_x8
    channel: 1
    rank: 2
  timing:
    preset: DDR4_2400R
  drampower_enable: true
  # power_debug: true  # option: debug log
  voltage:
    preset: Default  # option: voltage preset
  current:
    preset: Default  # option: current preset
```

```yaml
DRAM:
  impl: DDR4
  active_cycles_rank1: 9361
  pre_background_energy_rank1: 3850.9652475000003
  total_background_energy_rank0: 4424.0171849999997
  idle_cycles_rank1: 68489
  total_cmd_energy: 3203.0342735999993
  total_cmd_energy_rank0: 1601.8769927999997
  total_energy_rank0: 6025.8941777999989
  act_background_energy_rank0: 572.15229750000003
  pre_background_energy_rank0: 3851.8648874999999
  active_cycles_rank0: 9345
  act_background_energy_rank1: 573.13190550000002
  total_energy: 12051.1486116
  idle_cycles_rank0: 68505
  total_energy_rank1: 6025.2544338000007
  total_background_energy_rank1: 4424.0971530000006
  total_background_energy: 8848.1143379999994
  total_cmd_energy_rank1: 1601.1572807999996
AddrMapper:
  impl: RoBaRaCoCh
```

trace file result

❑ power는 어떻게 측정하느냐면

- power_debug option enable 시
- 오른쪽과 같은 power log 확인 가능

```yaml
DRAM:
  impl: DDR4
  org:
    preset: DDR4_8Gb_x8
    channel: 1
    rank: 2
  timing:
    preset: DDR4_2400R
  drampower_enable: true
  # power_debug: true  # option: debug log
  voltage:
    preset: Default  # option: voltage preset
  current:
    preset: Default  # option: current preset

```

```
root@947e591ed45c:/workspace# ./ramulator2 -f ./example_config_power_en.yaml
[Power] Rank0 ------ACT------ @ 19
[Power] Rank0 Rank is idle. idle_cycles: 19    active_start_cycle: 19 @ 19
[Power] Rank0 Bank0 Incrementing ACT counter. @ 19
[Power] Rank1 ------ACT------ @ 20
[Power] Rank1 Rank is idle. idle_cycles: 20    active_start_cycle: 20 @ 20
[Power] Rank1 Bank2 Incrementing ACT counter. @ 20
[Power] Rank0 ------ACT------ @ 23
[Power] Rank0 Bank3 Incrementing ACT counter. @ 23
[Power] Rank1 ------ACT------ @ 24
[Power] Rank1 Bank3 Incrementing ACT counter. @ 24
[Power] Rank0 Bank0 Incrementing RD counter. @ 35
[Power] Rank0 Bank3 Incrementing RD counter. @ 39
[Power] Rank0 Bank0 Incrementing RD counter. @ 43
[Power] Rank1 Bank2 Incrementing RD counter. @ 49
[Power] Rank1 Bank3 Incrementing RD counter. @ 53
[Power] Rank0 Bank0 Incrementing RD counter. @ 59
[Power] Rank0 ------PREA------ @ 9364
[Power] Rank0 Incrementing PRE counter. @ 9364
[Power] Rank0 Rank is not idle. active_cycles: 9345    idle_start_cycle: 9364 @ 9364
[Power] Rank0 ------REFab------ @ 9380
[Power] Rank0 Refresh starts. idle_cycles: 35 @ 9380
[Power] Rank1 ------PREA------ @ 9381
[Power] Rank1 Incrementing PRE counter. @ 9381
[Power] Rank1 Rank is not idle. active_cycles: 9361    idle_start_cycle: 9381 @ 9381
[Power] Rank1 ------REFab------ @ 9397
[Power] Rank1 Refresh starts. idle_cycles: 36 @ 9397
[Power] Rank0 ------REFab_end------ @ 9812
[Power] Rank0 Refresh ends. idle_start_cycle: 9812 @ 9812
[Power] Rank1 ------REFab_end------ @ 9829
[Power] Rank1 Refresh ends. idle_start_cycle: 9829 @ 9829
[Power] Rank0 ------REFab------ @ 18728
[Power] Rank0 Refresh starts. idle_cycles: 8951 @ 18728
[Power] Rank1 ------REFab------ @ 18729
[Power] Rank1 Refresh starts. idle_cycles: 8936 @ 18729
[Power] Rank0 ------REFab_end------ @ 19160
[Power] Rank0 Refresh ends. idle_start_cycle: 19160 @ 19160
[Power] Rank1 ------REFab_end------ @ 19161
[Power] Rank1 Refresh ends. idle_start_cycle: 19161 @ 19161
[Power] Rank0 ------REFab------ @ 28092
```

yaml/trace file for direct Read/Write Request

❑ example_config_dram.yaml & example_dram.trace & readwrite_trace.cpp

example_config_dram.yaml

```yaml
Frontend:
  impl: ReadWriteTrace
  path: example_dram.trace  # trace 파일 경로 (상대/절대
  clock_ratio: 1

  Translation:
    impl: RandomTranslation
    max_addr: 2147483648


MemorySystem:
  impl: GenericDRAM
  clock_ratio: 1

  DRAM:
    impl: DDR4
    org:
      preset: DDR4_8Gb_x8
```

example_dram.trace

```
R 0,0,0,0,0
W 0,0,0,0,128
R 0,0,0,1,0
W 0,0,0,1,256
R 0,0,1,0,0
W 0,0,1,0,512
R 0,0,0,1024,0
W 0,0,0,1024,0
```

```cpp
void tick() override {
  const Trace& t = m_trace[m_curr_trace_idx];
  m_memory_system->send({t.addr_vec, t.is_write ? Request::Type::Write : Request::Type::Read});
  // m_curr_trace_idx = (m_curr_trace_idx + 1) % m_trace_length;
  m_curr_trace_idx++; // revised by MinsungKim yonsei ISL lab - 2026.01.09.
};
```

- **Trace file을 arg로 받아, Frontend (ex: readwrite_trace.cpp)에서 처리되어 메모리 접근 request를 생성**

– **Read-Write Trace:** This frontend expects a trace file containing read (R) or store (W) requests with a *DRAM address vector* (i.e., which channel, rank, bank, row, column). Each line in the trace has the following format: **R/W ⟨comma-separated addr vector⟩:**

yaml/trace file for direct Read/Write Request

❑ Simulation Result

```
Frontend:
  impl: ReadWriteTrace

MemorySystem:
  impl: GenericDRAM
  total_num_other_requests: 0
  total_num_write_requests: 4
  total_num_read_requests: 4
  memory_system_cycles: 7
  DRAM:
    impl: DDR4
  AddrMapper:
    impl: RoBaRaCoCh
```

```
Controller:
  impl: Generic
  id: Channel 0
  avg_read_latency_0: 4.5
  read_queue_len_avg_0: 0.571428597
  write_queue_len_0: 12
  queue_len_0: 16
  num_other_reqs_0: 0
  num_write_reqs_0: 4
  read_latency_0: 18
  priority_queue_len_avg_0: 0
  row_hits_0: 0
  priority_queue_len_0: 0
  row_misses_0: 1
  row_conflicts_0: 0
  read_row_misses_0: 1
  queue_len_avg_0: 2.28571439
  read_row_conflicts_core_0: 0
  read_row_hits_0: 0
  write_queue_len_avg_0: 1.71428573
  read_row_conflicts_0: 0
  write_row_misses_0: 0
  write_row_conflicts_0: 0
  read_queue_len_0: 4
  write_row_hits_0: 0
  read_row_hits_core_0: 0
  read_row_misses_core_0: 0
  num_read_reqs_0: 4
  Scheduler:
    impl: FRFCFS
  RefreshManager:
    impl: AllBank


  RowPolicy:
    impl: ClosedRowPolicy
    num_close_reqs: 0
```

결과가 나오긴하는데 말이 안되는 수치를 가짐

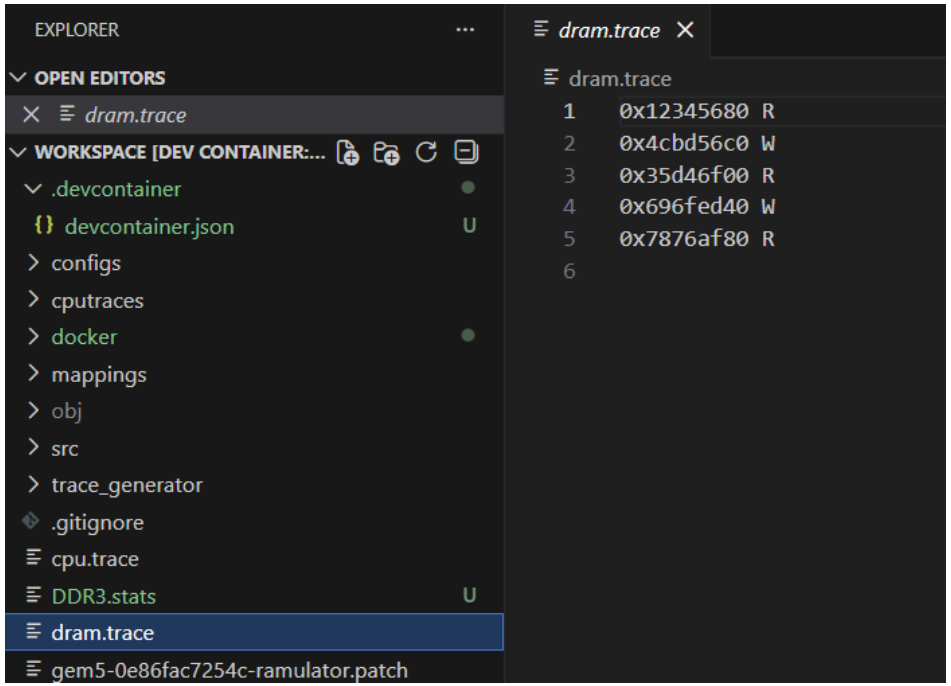
Currently, Ramulator 2 implements the following three frontends:

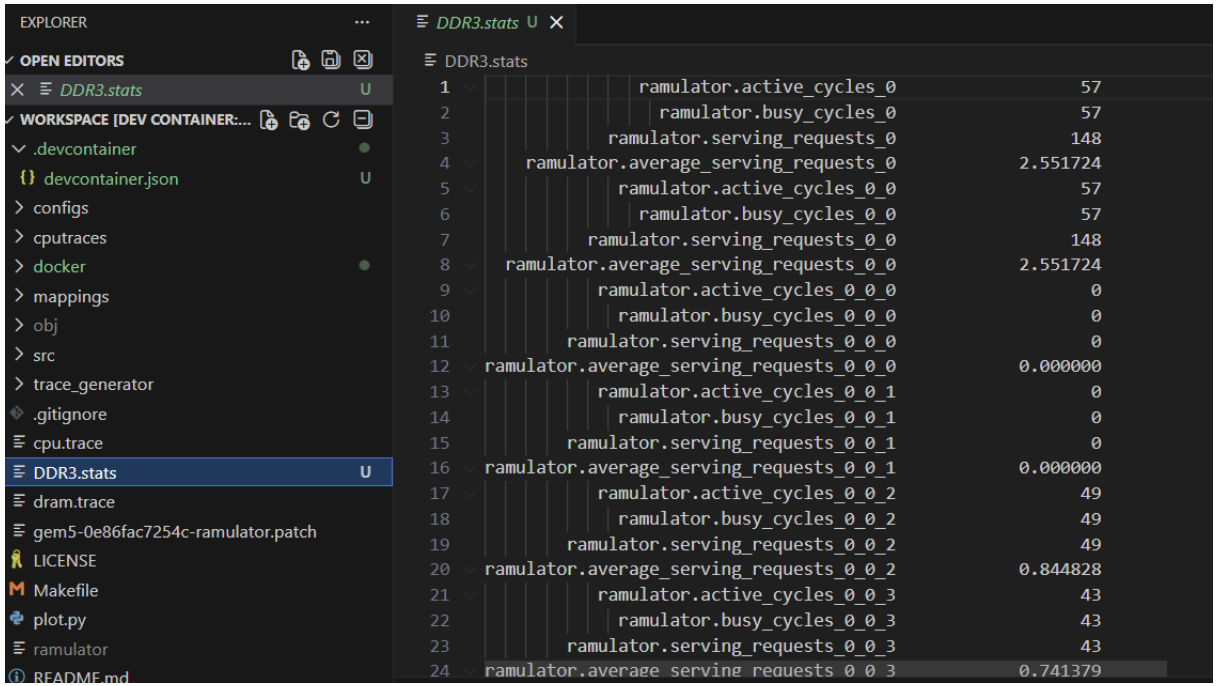
- **Memory Trace Frontend:** This frontend reads an input trace file that contains *main memory requests* of an application. It *sequentially* sends these requests to the memory system. This mode does not model any system in sufficient detail to perform timing simulations. Because of that, the Memory Trace Frontend is better suited for testing/debugging the functionality of newly added features. In the main tasks of this assignment, we will *not* use this frontend. Still, feel free to use this mode to test and debug your newly implemented features. Currently, Ramulator 2 implements the following two memory trace frontends:

- **Ramulator2.0은 Read/Write Request의 functionality만 검증**
- **Cycle Accurate하지 않음**

yaml/trace file for direct Read/Write Request

❑ Simulation Result of Ramulator1.0





Ramulator1.0은 납득할 수준의 결과가 나옴

Ramulator2.0의 전체 시스템을 끝까지 리뷰 후 Ramulator1.0의 DRAM subsystem만 simulation한 결과를 바탕으로 다시 분석 예정

src/frontend folder

❑ simpleO3 Core Operations

- class SimpleO3 – init()

```
28  void init() override {
29    m_clock_ratio = param<uint>("clock_ratio").required();
30
31    // Core params
32    std::vector<std::string> trace_list = param<std::vector<std::string>>("traces").desc("A list of traces.").required();
33    m_num_cores = trace_list.size();
34
35    int ipc   = param<int>("ipc").desc("IPC of the SimpleO3 core.").default_val(4);
36    int depth = param<int>("inst_window_depth").desc("Instruction window size of the SimpleO3 core.").default_val(128);
37
38    // LLC params
39    int llc_latency           = param<int>("llc_latency").desc("Aggregated latency of the LLC.").default_val(47);
40    int llc_linesize_bytes    = param<int>("llc_linesize").desc("LLC cache line size in bytes.").default_val(64);
41    int llc_associativity     = param<int>("llc_associativity").desc("LLC set associativity.").default_val(8);
42    int llc_capacity_per_core = parse_capacity_str(param<std::string>("llc_capacity_per_core").desc("LLC capacity per core.").default_val("2MB"));
43    int llc_num_mshr_per_core = param<int>("llc_num_mshr_per_core").desc("Number of LLC MSHR entries per core.").default_val(16);
44
45    // Simulation parameters
46    m_num_expected_insts = param<int>("num_expected_insts").desc("Number of instructions that the frontend should execute.").required();
47
48    // Create address translation module
49    m_translation = create_child_ifce<ITranslation>();
50
51    // Create the LLC
52    m_llc = new SimpleO3LLC(llc_latency, llc_capacity_per_core * m_num_cores, llc_linesize_bytes, llc_associativity, llc_num_mshr_per_core * m_num_cores);
53    // m_llc->deserialize(serialization_filename);
54    // m_llc->serialize(serialization_filename);
55
56    // Create the cores
57    for (int id = 0; id < m_num_cores; id++) {
58      SimpleO3Core* core = new SimpleO3Core(id, ipc, depth, m_num_expected_insts, trace_list[id], m_translation, m_llc);
59      core->m_callback = [this](Request& req){return this->receive(req);} ;
60      m_cores.push_back(core);
61    }
62
63    m_logger = Logging::create_logger("SimpleO3");
64
65    // Register the stats
66    register_stat(m_num_expected_insts).name("num_expected_insts");
67    register_stat(m_llc->s_llc_eviction).name("llc_eviction");
68    register_stat(m_llc->s_llc_read_access).name("llc_read_access");
69    register_stat(m_llc->s_llc_write_access).name("llc_write_access");
70    register_stat(m_llc->s_llc_read_misses).name("llc_read_misses");
71    register_stat(m_llc->s_llc_write_misses).name("llc_write_misses");
72    register_stat(m_llc->s_llc_mshr_unavailable).name("llc_mshr_unavailable");
73
74    for (int core_id = 0; core_id < m_cores.size(); core_id++) {
75      // register_stat(m_cores[core_id]->s_insts_retired).name("cycles_retired_core_{}", core_id);
76      register_stat(m_cores[core_id]->s_cycles_recorded).name("cycles_recorded_core_{}", core_id);
77      register_stat(m_cores[core_id]->s_mem_access_cycles).name("memory_access_cycles_recorded_core_{}", core_id);
78    }
79  }
80
```

1.1. Core Parameters를 yaml file에서 가져옴 → clock_ratio / traces (ipc, depth 하드 코딩)

1.2. LLC Parameters (latency, cache line size, 몇 assosiative cahce인지, MSHR 수, cache size) 하드 코딩

1.3. Address Translation / LLC / Cores(trace file의 수만큼 생성 됨) Instance 생성

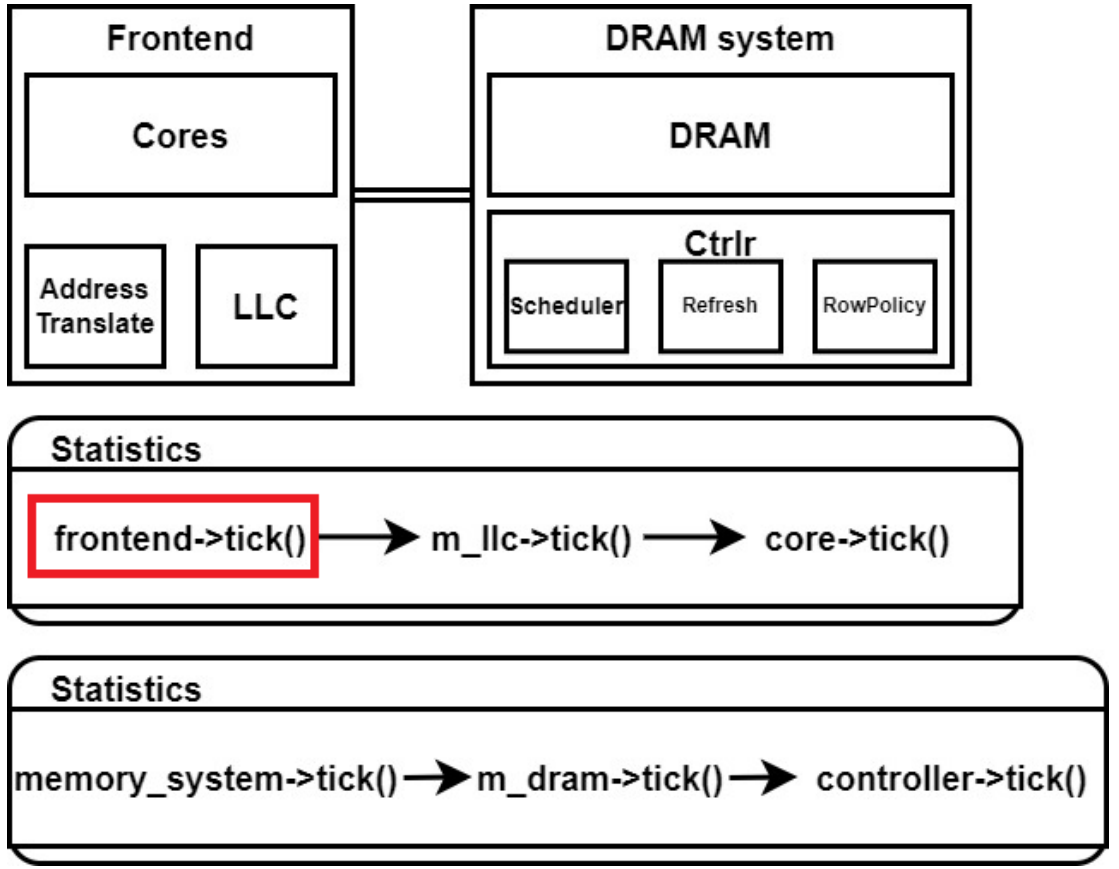
1.4. LLC의 동작 통계 변수들은 registration class에 등록

src/frontend folder

❑ simpleO3 Core Operations

- class SimpleO3 – tick() → frontend->tick()

```
81  void tick() override {
82    m_clk++;
83
84    if(m_clk % 10000000 == 0) {
85      m_logger->info("Processor Heartbeat {} cycles.", m_clk);
86    }
87
88    m_llc->tick();
89    for (auto core : m_cores) {
90      core->tick();
91    }
92  }
```



- m_clk (processor clk) 증가
- 아래의 순서로 frontend 동작
- Last Level Cache tick() → Core tick()

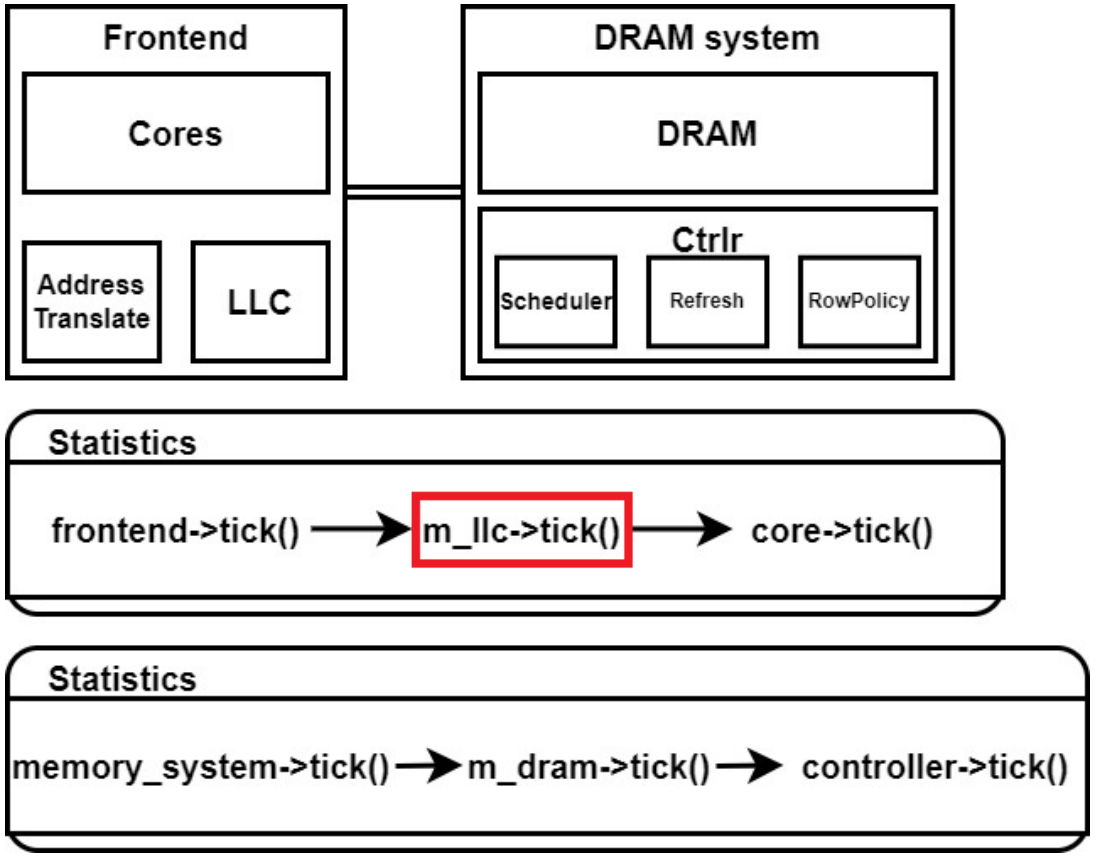
src/frontend folder

❑ simpleO3 Core Operations

- class SimpleO3 – tick() → m_llc->tick()

```cpp
void SimpleO3LLC::tick() {
  m_clk++;

  // Send miss requests to the memory system when LLC latency is met
  // TODO: Optimization by assuming in-order issue?
  auto it = m_miss_list.begin();
  while (it != m_miss_list.end()) {
    if (m_clk >= it->first) {
      if (!m_memory_system->send(it->second)) {
        it++;
      }
      else {
        it = m_miss_list.erase(it);
      }
    } else {
      it++;
    }
  }

  // call hit request callback when LLC latency is met
  it = m_hit_list.begin();
  while (it != m_hit_list.end()) {
    if (m_clk >= it->first) {
      std::vector<Request> _req_v{it->second};
      m_receive_requests[it->second.addr] = _req_v;

      it->second.callback(it->second);
      it = m_hit_list.erase(it);
    }
    else {
      it++;
    }
  }
};
```

```cpp
void SimpleO3LLC::evict_line(CacheSet_t& set, CacheSet_t::iterator victim_it) {
  DEBUG_LOG(DSIMPLEO3LLC, m_logger,  "Evicting {}.", victim_it->addr);
  s_llc_eviction++;

  // Generate writeback request if victim line is dirty
  if (victim_it->dirty) {
    Request writeback_req(victim_it->addr, Request::Type::Write);
    m_miss_list.push_back(std::make_pair(m_clk + m_latency, writeback_req));

    DEBUG_LOG(DSIMPLEO3LLC, m_logger,  "Writeback Request will be issued at Clk={}.", m_clk + m_latency);
  }

  set.erase(victim_it);
}
```



- Miss List에서 지연 끝난 Missed Request을 MemorySystem으로 전달
- LLC latency(m_latency) Simulation & Send missed req to DRAM
- 현재: LLC latency modeling 값 → m_latency = 47

src/frontend folder

❑ simpleO3 Core Operations

- class SimpleO3 – tick() → m_llc->tick()

```cpp
void SimpleO3LLC::tick() {
  m_clk++;

  // Send miss requests to the memory system when LLC latency is met
  // TODO: Optimization by assuming in-order issue?
  auto it = m_miss_list.begin();
  while (it != m_miss_list.end()) {
    if (m_clk >= it->first) {
      if (!m_memory_system->send(it->second)) {
        it++;
      }
      else {
        it = m_miss_list.erase(it);
      }
    } else {
      it++;
    }
  }

  // call hit request callback when LLC latency is met
  it = m_hit_list.begin();
  while (it != m_hit_list.end()) {
    if (m_clk >= it->first) {
      std::vector<Request> _req_v{it->second};
      m_receive_requests[it->second.addr] = _req_v;

      it->second.callback(it->second);
      it = m_hit_list.erase(it);
    }
    else {
      it++;
    }
  }
};
```

```cpp
if (auto line_it = check_set_hit(set, req.addr); line_it != set.end()) {
  // Hit in the set
  DEBUG_LOG(DSIMPLEO3LLC, m_logger,
  "[Clk={}] Request Source: {}, Type: {}, Addr: {}, Index: {}, Tag: {}. Hit, will finish at Clk={}",
  m_clk, req.source_id, req.type_id, req.addr, get_index(req.addr), get_tag(req.addr), m_clk, m_clk + m_latency
  );

  // Update the LRU status
  set.push_back({req.addr, get_tag(req.addr), line_it->dirty || (req.type_id == Request::Type::Write), true});
  set.erase(line_it);

  // Add to the hit list to callback when finished
  m_hit_list.push_back(std::make_pair(m_clk + m_latency, req));
  return true;
} else {
```

```cpp
void receive(Request& req) {
  m_llc->receive(req);

  // TODO: LLC latency for the core to receive the request?
  for (auto r : m_llc->m_receive_requests[req.addr]) {
    r.arrive = req.arrive;
    r.depart = req.depart;
    m_cores[r.source_id]->receive(r);
  }
  m_llc->m_receive_requests[req.addr].clear();
};
```

```cpp
// Create the cores
for (int id = 0; id < m_num_cores; id++) {
  SimpleO3Core* core = new SimpleO3Core(id, ipc, depth, m_num_expected_insts, trace_list
  core->m_callback = [this](Request& req){return this->receive(req);} ;
  m_cores.push_back(core);
}
```

- **Hit List에서 지연 끝난 Request들을 m_receive_requests배열에 저장**
  - 여러 Core가 같은 주소를 가지는 Requests를 기다리고 있을 때, 완료되면 모든 Core에게 통지할 수 있도록 Mapping 하는 것
  - ex. m_receive_requests[0x1000] = [Request(addr=0x1000, source_id=0)]
- **callback이 호출됨 (이 callback은 SimpleO3::receive를 가리킴)**

src/frontend folder

❑ simpleO3 Core Operations

- class SimpleO3 – tick() → m_llc->tick() → callback

```
56      // Create the cores
57      for (int id = 0; id < m_num_cores; id++) {
58        SimpleO3Core* core = new SimpleO3Core(id, ipc, depth, m_num_expected_insts, trace_list
59        core->m_callback = [this](Request& req){return this->receive(req);} ;
60        m_cores.push_back(core);
61      }
```

```
void receive(Request& req) {
  m_llc->receive(req);   → cache 내 MSLR 처리

  // TODO: LLC latency for the core to receive the request?
  for (auto r : m_llc->m_receive_requests[req.addr]) {
    r.arrive = req.arrive;
    r.depart = req.depart;
    m_cores[r.source_id]->receive(r);
  }
  m_llc->m_receive_requests[req.addr].clear();
};
```

```
184 void SimpleO3Core::receive(Request& req) {
185   m_window.set_ready(req.addr);
186
187   if (req.arrive != -1 && req.depart > m_last_mem_cycle) {
188     if (!reached_expected_num_insts) {
189       s_mem_access_cycles += (req.depart - std::max(m_last_mem_cycle, req.arrive));
190       m_last_mem_cycle = req.depart;
191     }
192   }
193 }
```

```
90  void SimpleO3Core::InstWindow::set_ready(Addr_t addr) {
91    if (m_load == 0) return;
92
93    int index = m_tail_idx;
94    for (int i = 0; i < m_load; i++) {
95      if (m_addr_list[index] == addr) {
96        m_ready_list[index] = true;
97      }
98      index++;
99      if (index == m_depth) {
100       index = 0;
101     }
102   }
103 }
```

Cycle 1: Instruction Issue → Cycle 2: Load 요청

Load addr=0x1000 → Request 생성 → LLC로 전송

Cycle 3-49: LLC latency (latency=47)

LLC: Hit 시 m_hit_list에 저장 후 latency 대기

Cycle 50: LLC Hit → callback

1. m_receive_requests[0x1000] = [Request(src=0)]
2. it->second.callback(it->second)
3. Simple03::receive() 실행
4. m_cores[0]->receive() 호출
5. m_window.set_ready(0x1000) 호출
6. ROB의 m_ready_list[idx] = true ← ready 표시

다음 core->tick() 시

1. m_window.retrieve()에서 이 Instruction 확인
2. ready_list[idx] == true이므로 Instruction Commit 가능
3. s_insts_retired++ (Commit된 Instructions 증가)
   이때, 현재 IPC = 4 이므로, 1cycle에 최대 4개 처리
   but 중간에 ready 아닌 것이 있으면 그 뒤는 모두 대기

- ROB에 쌓인 이미 완료된 Instruction은 이제 sequential하게 기존 Instruction 순서에 맞게 commit됨

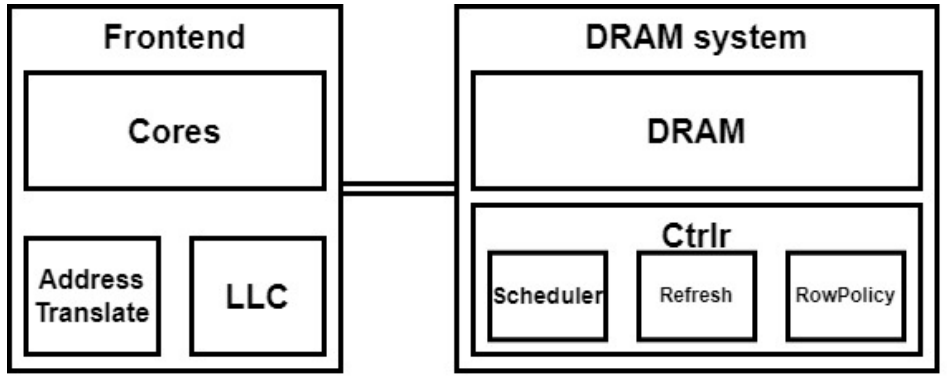
src/frontend folder

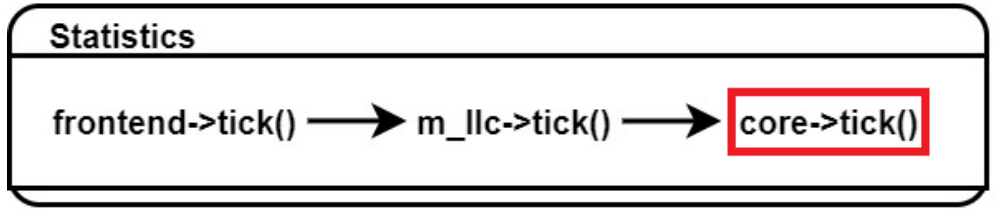
❑ simpleO3 Core Operations

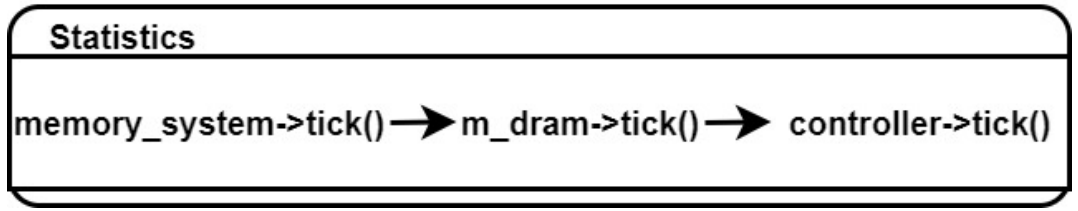
- class SimpleO3 – tick() → core->tick()

```
148  void SimpleO3Core::tick() {
149    m_clk++;
150
151    // 목표 명령어 개수 도달 시 시간 기록
152    s_insts_retired += m_window.retire();
153    if (!reached_expected_num_insts) {
154      if (s_insts_retired >= m_num_expected_insts) {
155        reached_expected_num_insts = true;
156        s_cycles_recorded = m_clk;
157      }
158    }
```

```
96  int SimpleO3Core::InstWindow::retire() {
97    if (m_load == 0) return 0;                    // ROB 비어있으면 0 반환
98
99    int num_retired = 0;
100   while (m_load > 0 && num_retired < m_ipc) {   // IPC 제한 내에서
101     if (!m_ready_list.at(m_tail_idx))           // tail이 준비 안 됐으면
102       break;                                    // 즉시 중단 (In-Order Retirement!)
103
104     m_tail_idx = (m_tail_idx + 1) % m_depth;    // tail 이동
105     m_load--;                                   // ROB 사용량 감소
106     num_retired++;                              // commit한 명령어 개수 증가
107   }
108   return num_retired;                           // 이 cycle에 commit된 명령어 수 반환
109 }
```







- ROB에서 명령어가 실행될 준비가 되면 ready flag를 **true**로 바꿈
- 이 경우 instruction이 실행되기 위한 data가 존재한다는 뜻
- 따라서, **ready = true**이면 해당 Inst를 commit 시킴 → Inst 처리됨

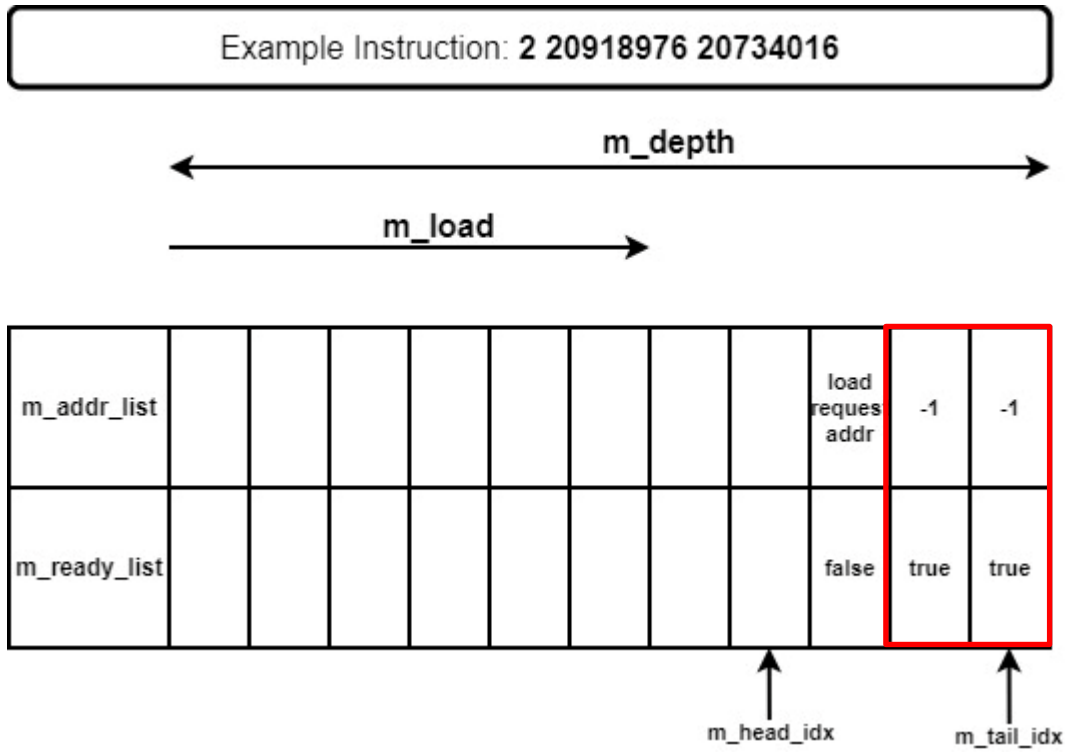
src/frontend folder

❑ simpleO3 Core Operations

- class SimpleO3 – tick() → core->tick()

```
160   // First, issue the non-memory instructions
161   int num_inserted_insts = 0;
162   while (m_num_bubbles > 0) {
163     if (num_inserted_insts == m_window.m_ipc) { // IPC 제한 도달
164       return;
165     }
166     if (m_window.is_full()) { // ROB 가득 참
167       return;
168     };
169     m_window.insert(true, -1); // ready=true (메모리 operation 아니기에)
170     num_inserted_insts++;
171     m_num_bubbles--;
172   }
```

```
80  // ROB에 새로운 명령어를 삽입
81  void SimpleO3Core::InstWindow::insert(bool ready, Addr_t addr) {
82    m_ready_list.at(m_head_idx) = ready;
83    m_addr_list.at(m_head_idx) = addr;
84
85    m_head_idx = (m_head_idx + 1) % m_depth;
86    m_load++;
87  }
```



- ROB에 Non-Memory Operation을 최대 IPC이하의 개수만큼 채워 넣음
- 위 Example Inst. (= 2 20918976 20734016)의 경우
  Non-Memory Op가 2개이기에 위 그림의 빨간 박스 부분이 완성됨

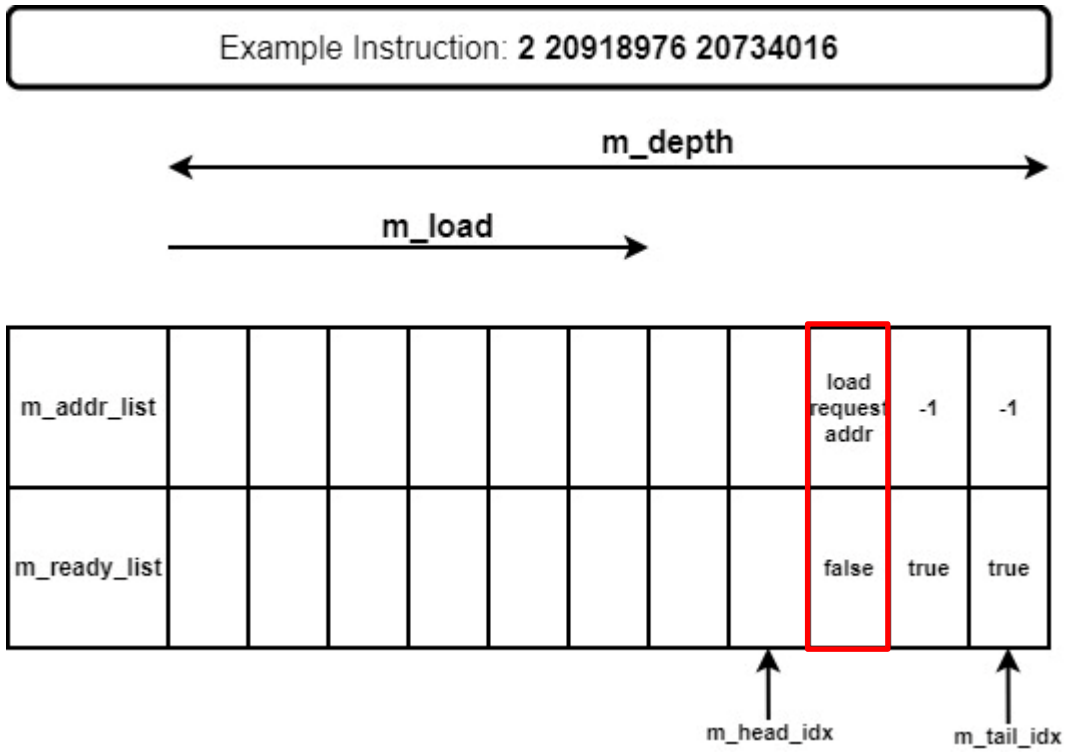
src/frontend folder

❑ simpleO3 Core Operations

- class SimpleO3 – tick() → core->tick()

```
174   // Second, try to send the load to the LLC
175   if (m_load_addr != -1) {
176     if (num_inserted_insts == m_window.m_ipc) { // IPC 제한
177       return;
178     }
179     if (m_window.is_full()) {                  // ROB 가득 참
180       return;
181     };
182
183     // Load 요청 생성 (가상 주소)
184     Request load_request(m_load_addr, Request::Type::Read, m_id, m_callback);
185
186     // 주소 변환 (가상 → 물리)
187     if (!m_translation->translate(load_request)) {
188       return; // 변환 실패
189     };
190
191     // LLC로 요청 발송
192     if (m_llc->send(load_request)) {
193       m_window.insert(false, load_request.addr);  // ready=false (메모리 대기)
194       m_load_addr = -1;                           // 처리 완료
195       if (m_writeback_addr != -1) {               // Store 있으면 다음 사이클에 발행
196         // If there is still writeback, return without getting the next trace line
197         // The write back will be issued in the next cycle
198         // TODO: Should we allow both load and writeback to issue at the same cycle?
199         return;
200       }
201     } else {
202       return;
203     }
204   }
```

ROB = InstWindow



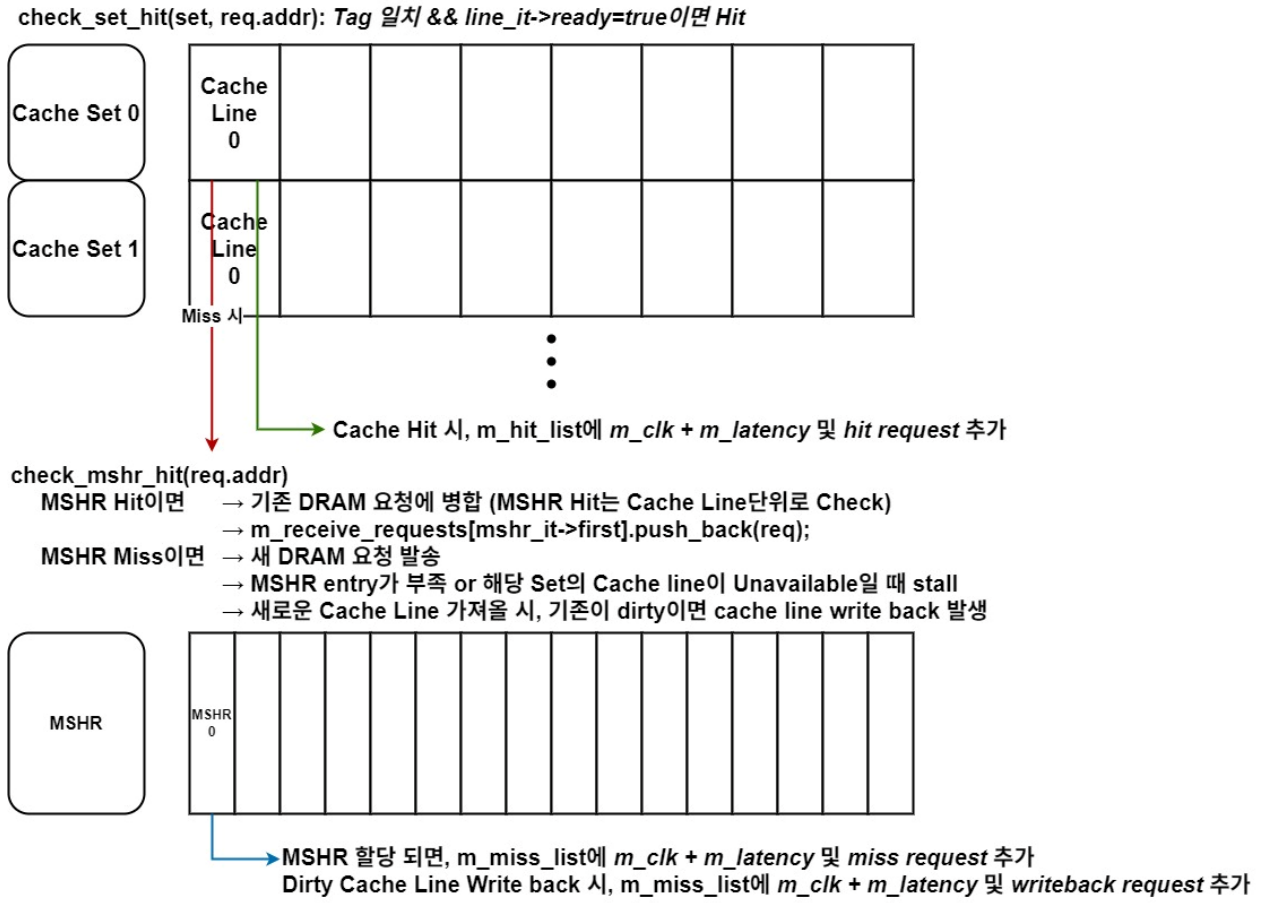
- Load Request가 존재하면, Load 요청의 주소를 translate
- 그 후, LLC로 Request를 보냄

src/frontend folder

❑ simpleO3 Core Operations

- class SimpleO3 – tick() → core->tick() → m_llc->send(load_request)

```
174   // Second, try to send the load to the LLC
175   if (m_load_addr != -1) {
176     if (num_inserted_insts == m_window.m_ipc) { // IPC 제한
177       return;
178     }
179     if (m_window.is_full()) {                  // ROB 가득 참
180       return;
181     };
182
183     // Load 요청 생성 (가상 주소)
184     Request load_request(m_load_addr, Request::Type::Read, m_id, m_callback);
185
186     // 주소 변환 (가상 → 물리)
187     if (!m_translation->translate(load_request)) {
188       return; // 변환 실패
189     };
190
191     // LLC로 요청 발송
192     if (m_llc->send(load_request)) {
193       m_window.insert(false, load_request.addr);  // ready=false (메모리 대기)
194       m_load_addr = -1;                           // 처리 완료
195       if (m_writeback_addr != -1) {               // Store 있으면 다음 사이클에 발행
196         // If there is still writeback, return without getting the next trace line
197         // The write back will be issued in the next cycle
198         // TODO: Should we allow both load and writeback to issue at the same cycle?
199         return;
200       }
201     } else {
202       return;
203     }
204   }
```



src/frontend folder

❑ simpleO3 Core Operations

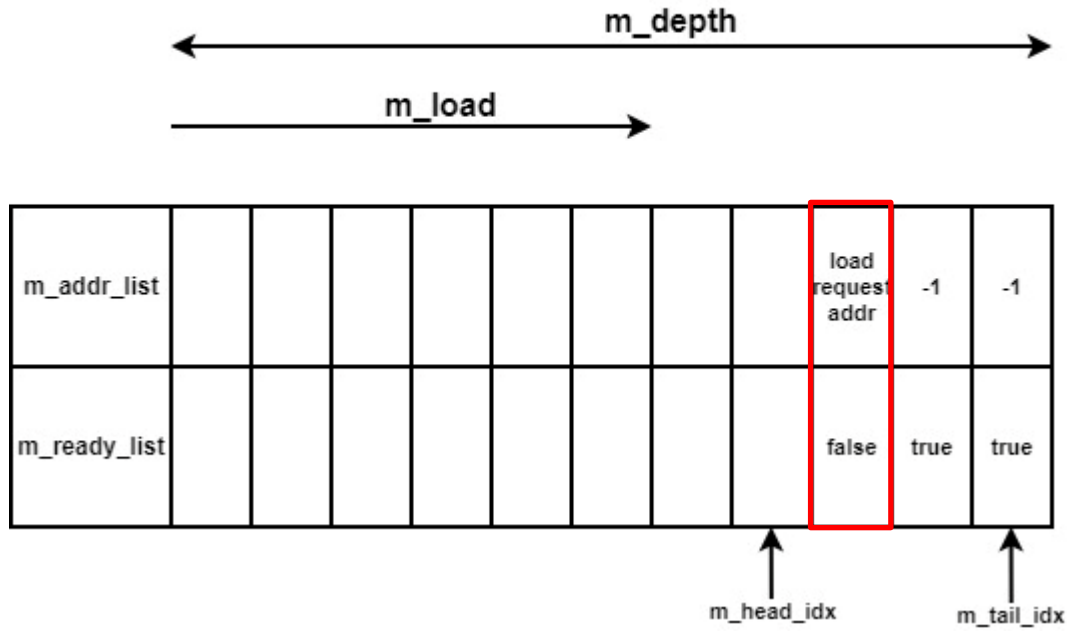
- class SimpleO3 – tick() → core->tick()

```
174   // Second, try to send the load to the LLC
175   if (m_load_addr != -1) {
176     if (num_inserted_insts == m_window.m_ipc) { // IPC 제한
177       return;
178     }
179     if (m_window.is_full()) {                  // ROB 가득 참
180       return;
181     };
182
183     // Load 요청 생성 (가상 주소)
184     Request load_request(m_load_addr, Request::Type::Read, m_id, m_callback);
185
186     // 주소 변환 (가상 → 물리)
187     if (!m_translation->translate(load_request)) {
188       return; // 변환 실패
189     };
190
191     // LLC로 요청 발송
192     if (m_llc->send(load_request)) {
193       m_window.insert(false, load_request.addr); // ready=false (메모리 대기)
194       m_load_addr = -1;                          // 처리 완료
195       if (m_writeback_addr != -1) {              // Store 있으면 다음 사이클에 발행
196         // If there is still writeback, return without getting the next trace line
197         // The write back will be issued in the next cycle
198         // TODO: Should we allow both load and writeback to issue at the same cycle?
199         return;
200       }
201     } else {
202       return;
203     }
204   }
```

```
80  // ROB에 새로운 명령어를 삽입
81  void SimpleO3Core::InstWindow::insert(bool ready, Addr_t addr) {
82    m_ready_list.at(m_head_idx) = ready;
83    m_addr_list.at(m_head_idx) = addr;
84
85    m_head_idx = (m_head_idx + 1) % m_depth;
86    m_load++;
87  }
```

ROB = InstWindow

Example Instruction: **2 20918976 20734016**



- LLC에 Request를 send하였으므로,
  해당 Load Request를 Instruction Window에 넣음
- Write Request가 존재 시, 이는 다음 tick()=cycle에 처리
- Write Request가 존재 시, return됨

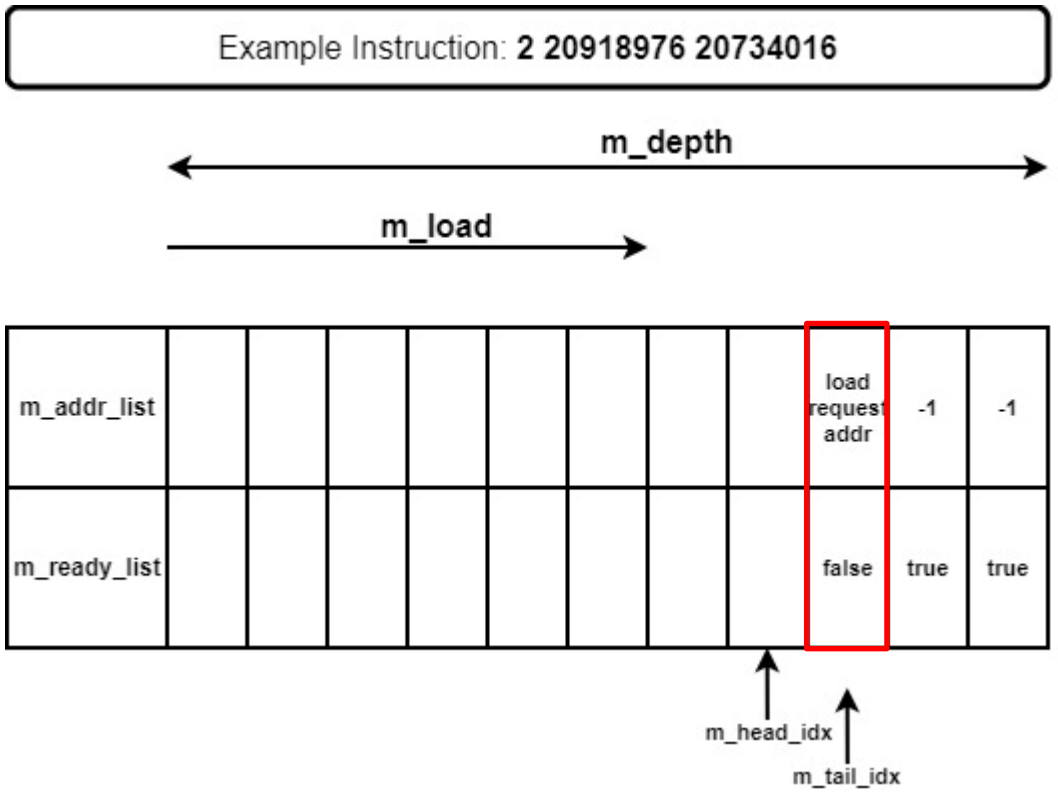
src/frontend folder

❑ simpleO3 Core Operations

- **class SimpleO3 – tick() → core->tick()**

```
206   // Third, try to send the writeback to the LLC
207   // Store(Writeback) 요청 발행
208   if (m_writeback_addr != -1) {
209     Request writeback_request(m_writeback_addr, Request::Type::Write, m_id, m_callback);
210     if (!m_translation->translate(writeback_request)) {
211       return;
212     };
213     if (!m_llc->send(writeback_request)) {
214       return;
215     }
216   }
217
218   // 다음 명령어 미리 로드
219   auto inst = m_trace.get_next_inst();
220   m_num_bubbles = inst.bubble_count;
221   m_load_addr = inst.load_addr;
222   m_writeback_addr = inst.store_addr;
223 }
```

ROB = InstWindow



- Write Request가 존재 시, return되었기에, 아래 다음 Instruction load되지 않고, 진행되고 있던 Inst 처리
- core->tick()에서 *1. issue the non-memory instructions / 2. try to send the load to the LLC* 건너뛰고 Write 처리
- Write Request도 Read와 마찬가지로, LLC의 send()함수를 실행시킨다.
- 그 후, 다음 Instruction의 정보를 core.h에 update한다.
- 또한, Write Request는 CPU가 issue후 신경 안 써도 되니, ROB에 저장하지 않는다.

src/memory_system folder

❑ memory_system

- class SimpleO3 – tick() → m_llc->tick() → m_memory_system – send()

```
37  void SimpleO3LLC::tick() {
38    m_clk++;
39
40    // Send miss requests to the memory system when LLC latency is met
41    // TODO: Optimization by assuming in-order issue?
42    auto it = m_miss_list.begin();
43    while (it != m_miss_list.end()) {
44      if (m_clk >= it->first) {                    // 지연 완료되었는가?
45        if (!m_memory_system->send(it->second)) { // 발송 실패 → 다음 사이클 재시도
46          it++;
47        }
48        else {
49          it = m_miss_list.erase(it);              // 발송 성공 → 제거
50        }
51      } else {
52        it++;                                      // 아직 시간 미도달
53      }
54    }
```

```
50  bool send(Request req) override {
51    // seperate physical address to dram vector, [channel, rank, bank, ...]
52    m_addr_mapper->apply(req);
53    int channel_id = req.addr_vec[0];                    // 채널 ID 추출
54    bool is_success = m_controllers[channel_id]->send(req); // 해당 채널 컨트롤러로 요청 전송
55
56    // update statisticss
57    if (is_success) {
58      switch (req.type_id) {
59        case Request::Type::Read: {
60          s_num_read_requests++;
61          break;
62        }
63        case Request::Type::Write: {
64          s_num_write_requests++;
65          break;
66        }
67        default: {
68          s_num_other_requests++;
69          break;
70        }
71      }
72    }
73
74    return is_success;
75  };
```

- Frontend에서 ***LLC miss가 발생 시***, 해당 request들을 m_miss_list에 queuing하고,
- cache latency가 끝나면, ***m_miss_list의 request들을 memory_system으로 send***
- memory_system은 physical address → dram vector 후 ***알맞은 channel의 ctrlr으로 send***

src/memory_system folder

❑ memory_system

- m_memory_system – send() → m_controllers[channel_id] – send()

```
50  bool send(Request req) override {
51    // seperate physical address to dram vector, [channel, rank, bank, ...]
52    m_addr_mapper->apply(req);
53    int channel_id = req.addr_vec[0];                    // 채널 ID 추출
54    bool is_success = m_controllers[channel_id]->send(req); // 해당 채널 컨트롤러로 요청 전송
55
56    // update statisticss
57    if (is_success) {
58      switch (req.type_id) {
59        case Request::Type::Read: {
60          s_num_read_requests++;
61          break;
62        }
63        case Request::Type::Write: {
64          s_num_write_requests++;
65          break;
66        }
67        default: {
68          s_num_other_requests++;
69          break;
70        }
71      }
72    }
73
74    return is_success;
75  };
```

- 각 request의 final command를 지정
- Requests의 통계 update

m_controllers[channel_id] – send() 처음 부분

```
115   bool send(Request& req) override {
116     // Request type별 최종 커맨드 결정
117     req.final_command = m_dram->m_request_translations(req.type_id);
118
119     switch (req.type_id) {
120       case Request::Type::Read: {
121         s_num_read_reqs++;
122         break;
123       }
124       case Request::Type::Write: {
125         s_num_write_reqs++;
126         break;
127       }
128       default: {
129         s_num_other_reqs++;
130         break;
131       }
132     }
133
```

```
112     inline static const ImplLUT m_request_translations = LUT (
113       m_requests, m_commands, {
114         {"read", "RD"}, {"write", "WR"}, {"all-bank-refresh", "REFab"},
115         {"open-row", "ACT"}, {"close-row", "PRE"}
116       }
117     );
```

⋮

```
147     // Else, enqueue them to corresponding buffer based on request type id
148     bool is_success = false;
149     req.arrive = m_clk;                                  // 도착 시간 기록
150     if        (req.type_id == Request::Type::Read) {
151       is_success = m_read_buffer.enqueue(req);
152     } else if (req.type_id == Request::Type::Write) {
153       is_success = m_write_buffer.enqueue(req);
154     } else {
155       throw std::runtime_error("Invalid request type!");
156     }
157     if (!is_success) {
158       // We could not enqueue the request
159       req.arrive = -1;
160       return false;
161     }
162
163     return true;
164   };
```

src/memory_system folder

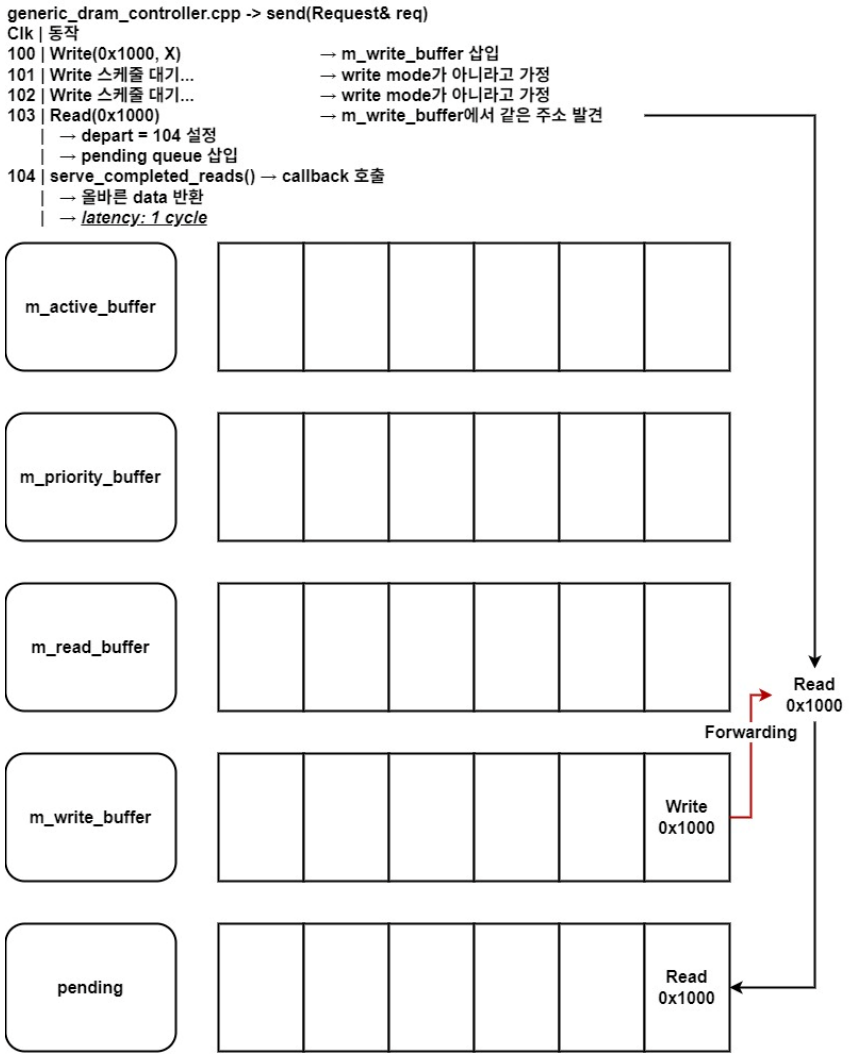
❑ memory_system

- M_memory_system – send() → m_controllers[channel_id] – send()

m_controllers[channel_id] – send() continued ...

```
134        // Forward existing write requests to incoming read requests
135        if (req.type_id == Request::Type::Read) {
136          auto compare_addr = [req](const Request& wreq) {
137            return wreq.addr == req.addr;
138          };
139          if (std::find_if(m_write_buffer.begin(), m_write_buffer.end(), compare_addr) != m_write_buffer.end()) {
140            // The request will depart at the next cycle
141            req.depart = m_clk + 1;
142            pending.push_back(req); // DRAM 접근 없이 반환 -> 다음 사이클에 데이터 전달
143            return true;
144          }
145        }
```

- Read Request가 들어왔을 시,
- 기존 write buffer에 Read Request와 같은 주소를 가지는 Req 존재 시,
- Data Forwarding 진행
- Read요청이 원하는 Data를 forwarding을 통해 얻었음.
- 이 Read Request를 Pending Queue에 넣음



src/memory_system folder

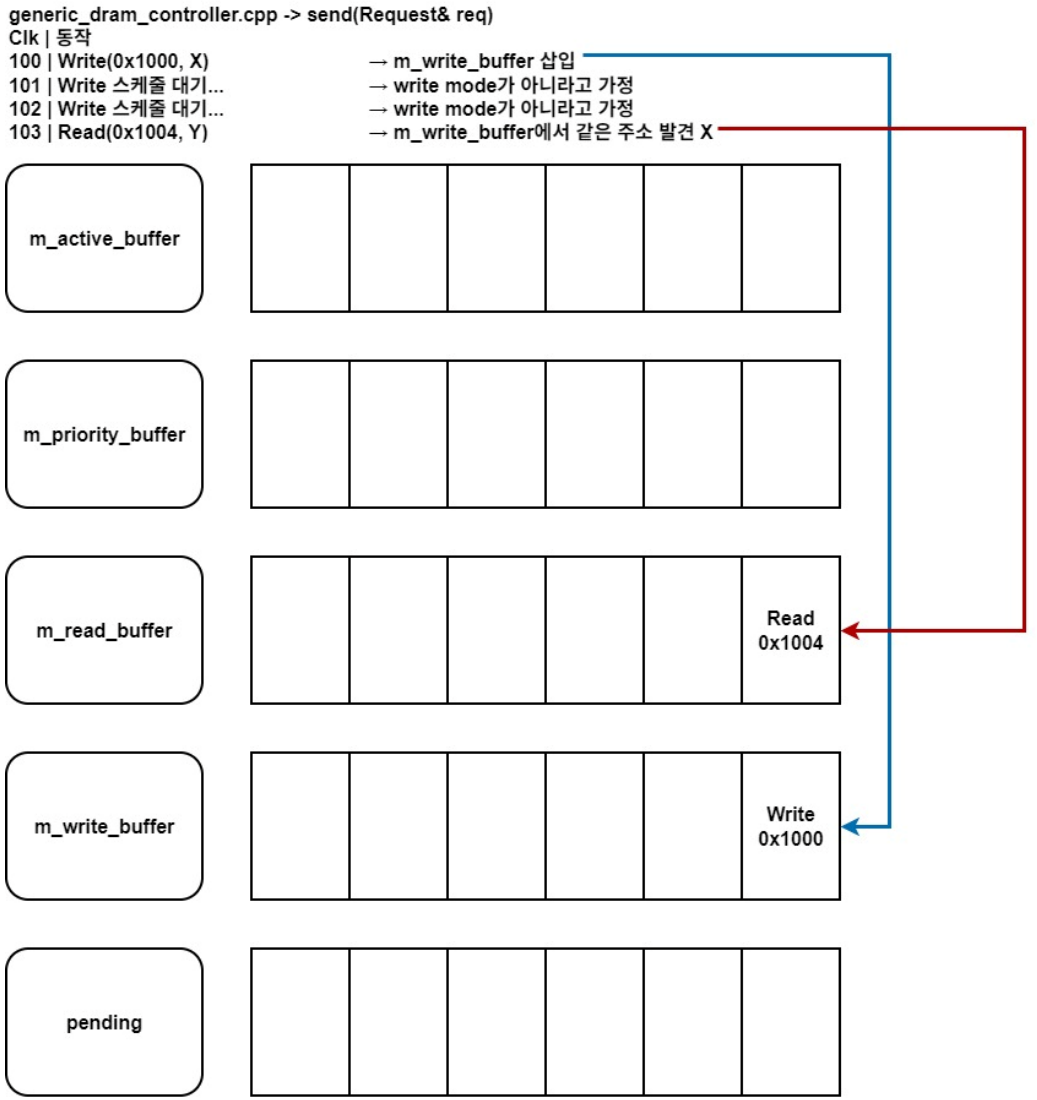
❑ memory_system

- m_memory_system – send() → m_controllers[channel_id] – send()

m_controllers[channel_id] – send() continued …

```
147     // Else, enqueue them to corresponding buffer based on request type id
148     bool is_success = false;
149     req.arrive = m_clk;                                  // 도착 시간 기록
150     if        (req.type_id == Request::Type::Read) {
151       is_success = m_read_buffer.enqueue(req);
152     } else if (req.type_id == Request::Type::Write) {
153       is_success = m_write_buffer.enqueue(req);
154     } else {
155       throw std::runtime_error("Invalid request type!");
156     }
157     if (!is_success) {
158       // We could not enqueue the request
159       req.arrive = -1;
160       return false;
161     }
```

- Data Forwarding 진행할 수 있는 조건 X 시,
- Request Type에 맞춰서 해당하는 buffer에 넣음



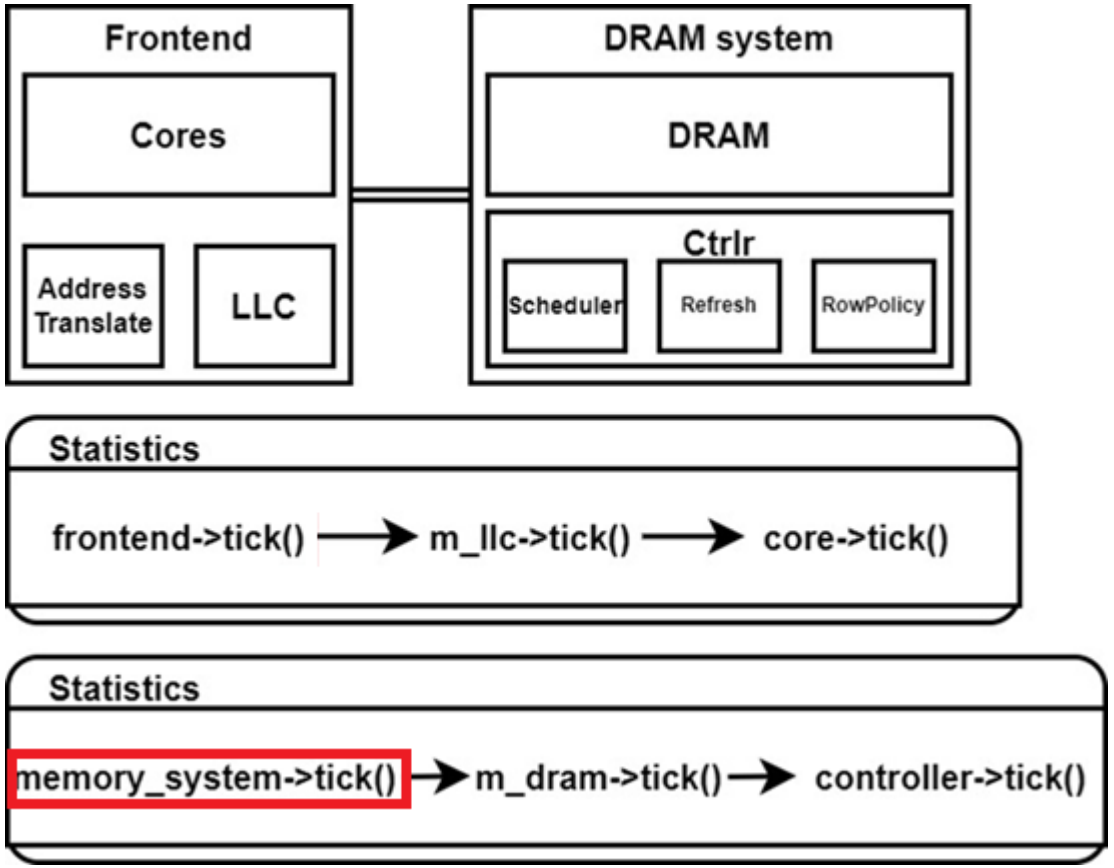
src/memory_system folder

❑ memory_system

- m_memory_system – tick()

```
77    void tick() override {
78      m_clk++;
79      m_dram->tick();          // DRAM 내부 상태 업데이트 (리프레시, 데이터 전송 등)
80      for (auto controller : m_controllers) {
81        controller->tick();    // 요청 스케줄링, 커맨드 발행, 완료 처리
82      }
83    };
```

- m_clk (memory clk) 증가
- 아래의 순서로 Memory System 동작
- Dram tick() → Controller tick()



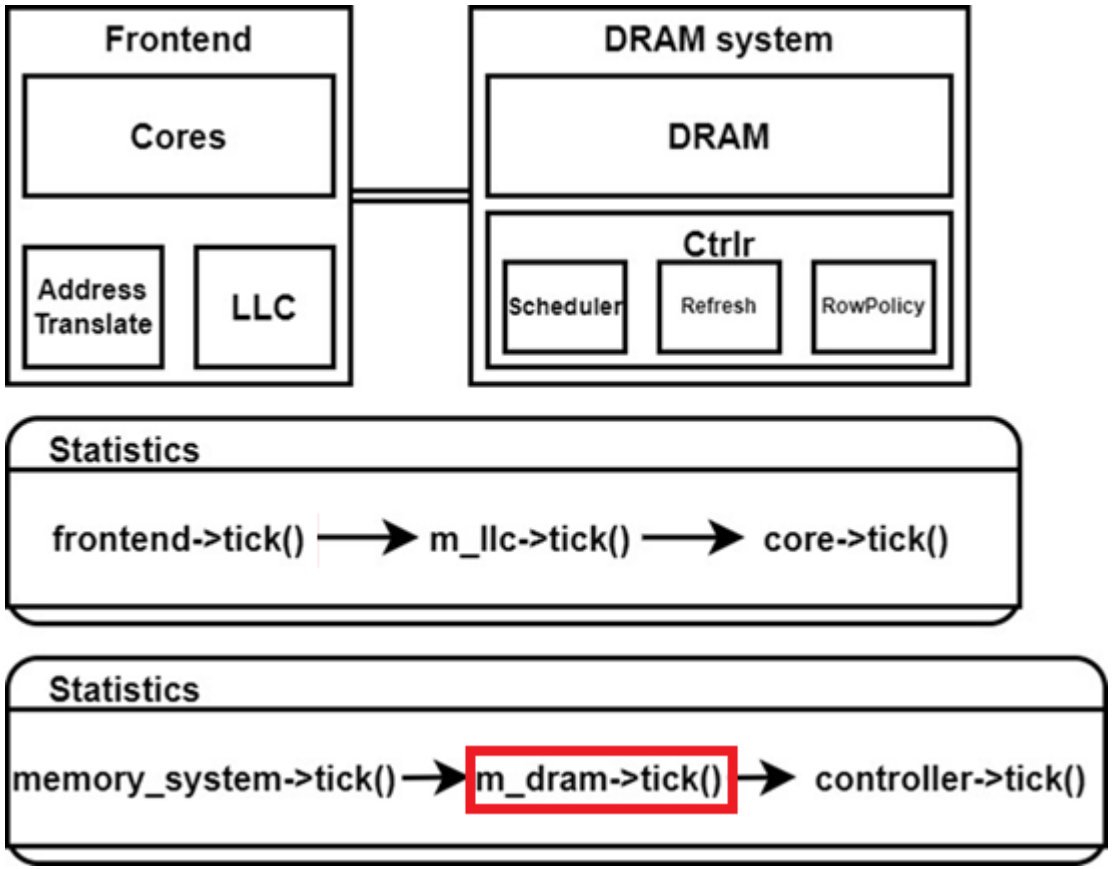
src/memory_system folder

❑ memory_system

- m_memory_system – tick() → m_dram – tick()

```
183   void tick() override {
184     m_clk++;
185
186     // Check if there is any future action at this cycle
187     for (int i = m_future_actions.size() - 1; i >= 0; i--) {
188       auto& future_action = m_future_actions[i];
189       if (future_action.clk == m_clk) {
190         handle_future_action(future_action.cmd, future_action.addr_vec);
191         m_future_actions.erase(m_future_actions.begin() + i);
192       }
193     }
194   };
```

```
232   void handle_future_action(int command, const AddrVec_t& addr_vec) {
233     int channel_id = addr_vec[m_levels["channel"]];
234     switch (command) {
235       case m_commands("REFab"):
236         m_channels[channel_id]->update_powers(m_commands("REFab_end"), addr_vec, m_clk);
237         m_channels[channel_id]->update_states(m_commands("REFab_end"), addr_vec, m_clk);
238         break;
239       default:
240         // Other commands do not require future actions
241         break;
242     }
243   };
```



- future_action 배열은 Dram Controller에서 DRAM으로 *REFab command*를 issue 시 push_back됨
- m_dram->tick() 시에는 이 future_action 배열을 확인하고,
- timing constraints를 만족하는 clk 때, REFab_end를 issue.
- REFab가 issue되고, 해당 Rank가 Refreshing되는 동안 해당 Rank는 다른 command issue 불가

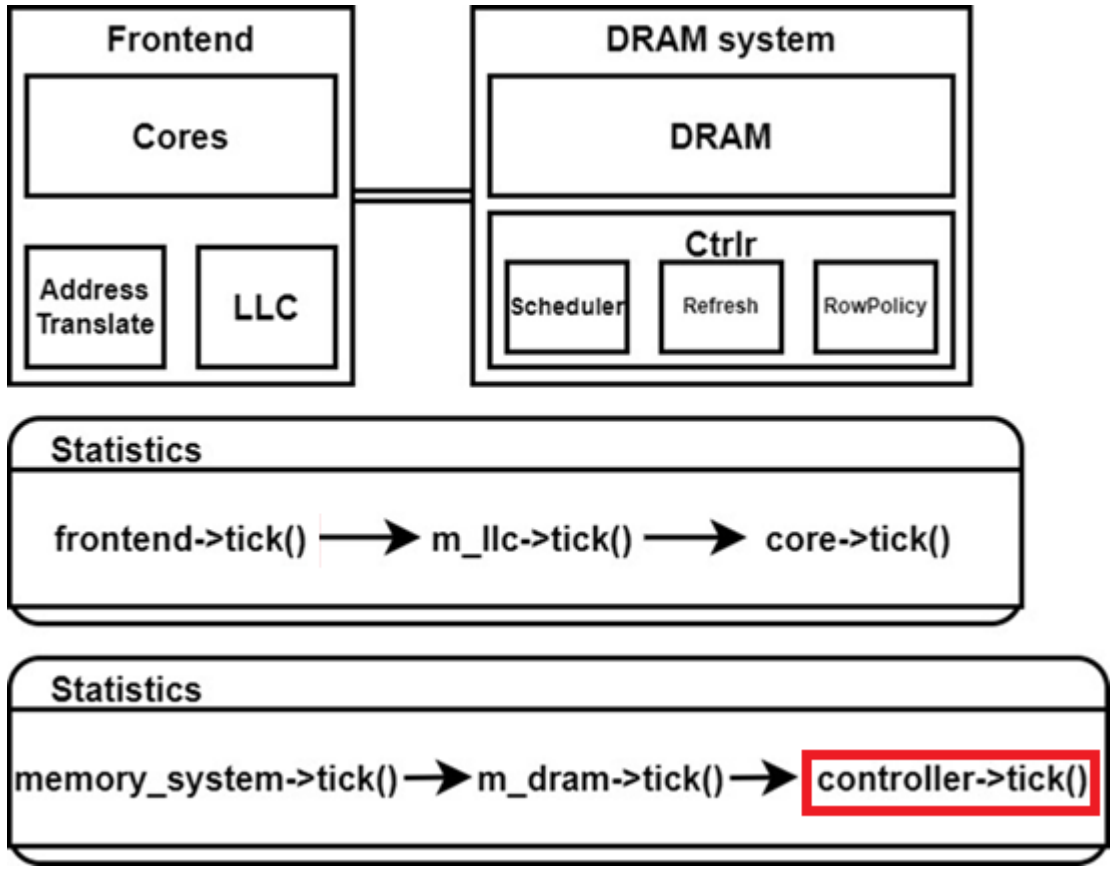
src/memory_system folder

❑ memory_system

- m_memory_system – tick()

```
77    void tick() override {
78      m_clk++;
79      m_dram->tick();          // DRAM 내부 상태 업데이트 (리프레시, 데이터 전송 등)
80      for (auto controller : m_controllers) {
81        controller->tick();    // 요청 스케줄링, 커맨드 발행, 완료 처리
82      }
83    };
```

- m_clk (memory clk) 증가
- 아래의 순서로 Memory System 동작
- Dram tick() → Controller tick()
- Controller는 서로 다른 channel마다 존재



src/memory_system folder

❑ memory_system

- m_memory_system – tick() → m_controllers[channel_id] – tick()

```cpp
void tick() override {
  m_clk++;

  // Update statistics
  s_queue_len += m_read_buffer.size() + m_write_buffer.size() + m_priority_buffer.size() + pending.size();
  s_read_queue_len += m_read_buffer.size() + pending.size();
  s_write_queue_len += m_write_buffer.size();
  s_priority_queue_len += m_priority_buffer.size();

  // 1. Serve completed reads - 완료된 읽기 요청 처리
  serve_completed_reads();

  m_refresh->tick();  // 리프레시 관리

  // 2. Try to find a request to serve.- 스케줄링 (어떤 요청을 발행할지 결정)
  ReqBuffer::iterator req_it;
  ReqBuffer* buffer = nullptr;
  bool request_found = schedule_request(req_it, buffer);

  // 2.1 Take row policy action
  m_rowpolicy->update(request_found, req_it);

  // 3. Update all plugins
  for (auto plugin : m_plugins) {
    plugin->update(request_found, req_it);
  }
```

generic_dram_controller.cpp

```cpp
// 4. Finally, issue the commands to serve the request
if (request_found) {
  // If we find a real request to serve
  if (req_it->is_stat_updated == false) {
    update_request_stats(req_it); // 히트/미스 통계
  }
  m_dram->issue_command(req_it->command, req_it->addr_vec);

  // If we are issuing the last command, set depart clock cycle and move the request to the pending queue
  if (req_it->command == req_it->final_command) {
    if (req_it->type_id == Request::Type::Read) {
      req_it->depart = m_clk + m_dram->m_read_latency;
      pending.push_back(*req_it);
    } else if (req_it->type_id == Request::Type::Write) {
      // TODO: Add code to update statistics
    }
    buffer->remove(req_it);
  } else {
    if (m_dr
      if (m_
        bu
      }
    }
  }

}
};
```

```cpp
void serve_completed_reads() {
  if (pending.size()) {
    // Check the first pending request
    auto& req = pending[0];
    if (req.depart <= m_clk) {  // DRAM에서 데이터 도착했는가?
      // Request received data from dram
      if (req.depart - req.arrive > 1) {  // forwarding 외 읽기 레이턴시 집계 (forwarding = 1cycle latency)
        // Check if this requests accesses the DRAM or is being forwarded.
        // TODO add the stats back
        s_read_latency += req.depart - req.arrive;
      }

      // callback 호출 (LLC의 receive() 함수)
      if (req.callback) {
        // If the request comes from outside (e.g., processor), call its callback
        req.callback(req);  // <- LLC로 데이터 전달
      }
      // Finally, remove this request from the pending queue
      pending.pop_front();  // queue에서 제거
    }
  };
};
```

1. 먼저 pending queue[0]에서 depart시간 constraint를 만족 시킨 read request를 완료 처리 함

src/memory_system folder

❑ memory_system

- m_memory_system – tick() → m_controllers[channel_id] – tick()

```cpp
void tick() override {
  m_clk++;

  // Update statistics
  s_queue_len += m_read_buffer.size() + m_write_buffer.size() + m_priority_buffer.size() + pending.size();
  s_read_queue_len += m_read_buffer.size() + pending.size();
  s_write_queue_len += m_write_buffer.size();
  s_priority_queue_len += m_priority_buffer.size();

  // 1. Serve completed reads - 완료된 읽기 요청 처리
  serve_completed_reads();

  m_refresh->tick();  // 리프레시 관리

  // 2. Try to find a request to serve.- 스케줄링 (어떤 요청을 발행할지 결정)
  ReqBuffer::iterator req_it;
  ReqBuffer* buffer = nullptr;
  bool request_found = schedule_request(req_it, buffer);

  // 2.1 Take row policy action
  m_rowpolicy->update(request_found, req_it);

  // 3. Update all plugins
  for (auto plugin : m_plugins) {
    plugin->update(request_found, req_it);
  }

  // 4. Finally, issue the commands to serve the request
  if (request_found) {
    // If we find a real request to serve
    if (req_it->is_stat_updated == false) {
      update_request_stats(req_it); // 히트/미스 통계
    }
    m_dram->issue_command(req_it->command, req_it->addr_vec);

    // If we are issuing the last command, set depart clock cycle and move the request to the pending queue
    if (req_it->command == req_it->final_command) {
      if (req_it->type_id == Request::Type::Read) {
        req_it->depart = m_clk + m_dram->m_read_latency;
        pending.push_back(*req_it);
      } else if (req_it->type_id == Request::Type::Write) {
        // TODO: Add code to update statistics
      }
      buffer->remove(req_it);
    } else {
      if (m_dram->m_command_meta(req_it->command).is_opening) {
        if (m_active_b
          buffer->remo
        }
      }
    }

  }
};
```

all_bank_refresh.cpp – tick()

```cpp
void tick() {
  m_clk++;

  if (m_clk == m_next_refresh_cycle) {
    m_next_refresh_cycle += m_nrefi;
    for (int r = 0; r < m_num_ranks; r++) {
      std::vector<int> addr_vec(m_dram_org_levels, -1);
      addr_vec[0] = m_ctrl->m_channel_id;
      addr_vec[1] = r;
      Request req(addr_vec, m_ref_req_id);

      bool is_success = m_ctrl->priority_send(req);
      if (!is_success) {
        throw std::runtime_error("Failed to send refresh!");
      }
    }
  }
};
```

```cpp
// tREFI(base) table (unit is nanosecond!)
constexpr int tREFI_BASE = 7800;
```

2. Refresh는 next_refresh_cycle에 도달할 때 마다,
   모든 Rank에 Refresh 요청 발행
   이때, priority queue에 REFab cmd를 넣음

src/memory_system folder

❑ memory_system

- m_memory_system – tick() → m_controllers[channel_id] – tick()

```cpp
void tick() override {
  m_clk++;

  // Update statistics
  s_queue_len += m_read_buffer.size() + m_write_buffer.size() + m_priority_buffe
  s_read_queue_len += m_read_buffer.size() + pending.size();
  s_write_queue_len += m_write_buffer.size();
  s_priority_queue_len += m_priority_buffer.size();

  // 1. Serve completed reads - 완료된 읽기 요청 처리
  serve_completed_reads();

  m_refresh->tick();  // 리프레시 관리

  // 2. Try to find a request to serve.- 스케줄링 (어떤 요청을 발행할지 결정)
  ReqBuffer::iterator req_it;
  ReqBuffer* buffer = nullptr;
  bool request_found = schedule_request(req_it, buffer);

  // 2.1 Take row policy action
  m_rowpolicy->update(request_found, req_it);

  // 3. Update all plugins
  for (auto plugin : m_plugins) {
    plugin->update(request_found, req_it);
  }
```

schedule_request

```cpp
bool schedule_request(ReqBuffer::iterator& req_it, ReqBuffer*& req_buffer) {
  bool request_found = false;
  // 2.1    First, check the act buffer to serve requests that are already activating (avoid useless ACTs)
  if (req_it= m_scheduler->get_best_request(m_active_buffer); req_it != m_active_buffer.end()) {
    if (m_dram->check_ready(req_it->command, req_it->addr_vec)) {
      request_found = true;
      req_buffer = &m_active_buffer;
    }
  }

  // 2.2    If no requests can be scheduled from the act buffer, check the rest of the buffers
  if (!request_found) {
    // 2.2.1    We first check the priority buffer to prioritize e.g., maintenance requests
    if (m_priority_buffer.size() != 0) {
      req_buffer = &m_priority_buffer;
      req_it = m_priority_buffer.begin();
      req_it->command = m_dram->get_preq_command(req_it->final_command, req_it->addr_vec);

      request_found = m_dram->check_ready(req_it->command, req_it->addr_vec);
      if (!request_found & m_priority_buffer.size() != 0) {
        return false;
      }
    }

    // 2.2.1    If no request to be scheduled in the priority buffer, check the read and write buffers.
    if (!request_found) {
      // Query the write policy to decide which buffer to serve
      set_write_mode();
      auto& buffer = m_is_write_mode ? m_write_buffer : m_read_buffer;
      if (req_it = m_scheduler->get_best_request(buffer); req_it != buffer.end()) {
        request_found = m_dram->check_ready(req_it->command, req_it->addr_vec);
        req_buffer = &buffer;
      }
    }
  }

  // 2.3 If we find a request to schedule, we need to check if it will close an opened row in the active buffer.
  if (request_found) {
    if (m_dram->m_command_meta(req_it->command).is_closing) {
      auto& rowgroup = req_it->addr_vec;
      for (auto _it = m_active_buffer.begin(); _it != m_active_buffer.end(); _it++) {
        auto& _it_rowgroup = _it->addr_vec;
        bool is_matching = true;
        for (int i = 0; i < m_bank_addr_idx + 1 ; i++) {
          if (_it_rowgroup[i] != rowgroup[i] && _it_rowgroup[i] != -1 && rowgroup[i] != -1) {
            is_matching = false;
            break;
          }
        }
        if (is_matching) {
          request_found = false;
          break;
        }
      }
    }

  return request_found;
}
```

```
    }

  }

};
```

3. Request들을 Scheduling하여 적절한 Request Buffer를 buffer에 넣고, iteration pointer도 반환

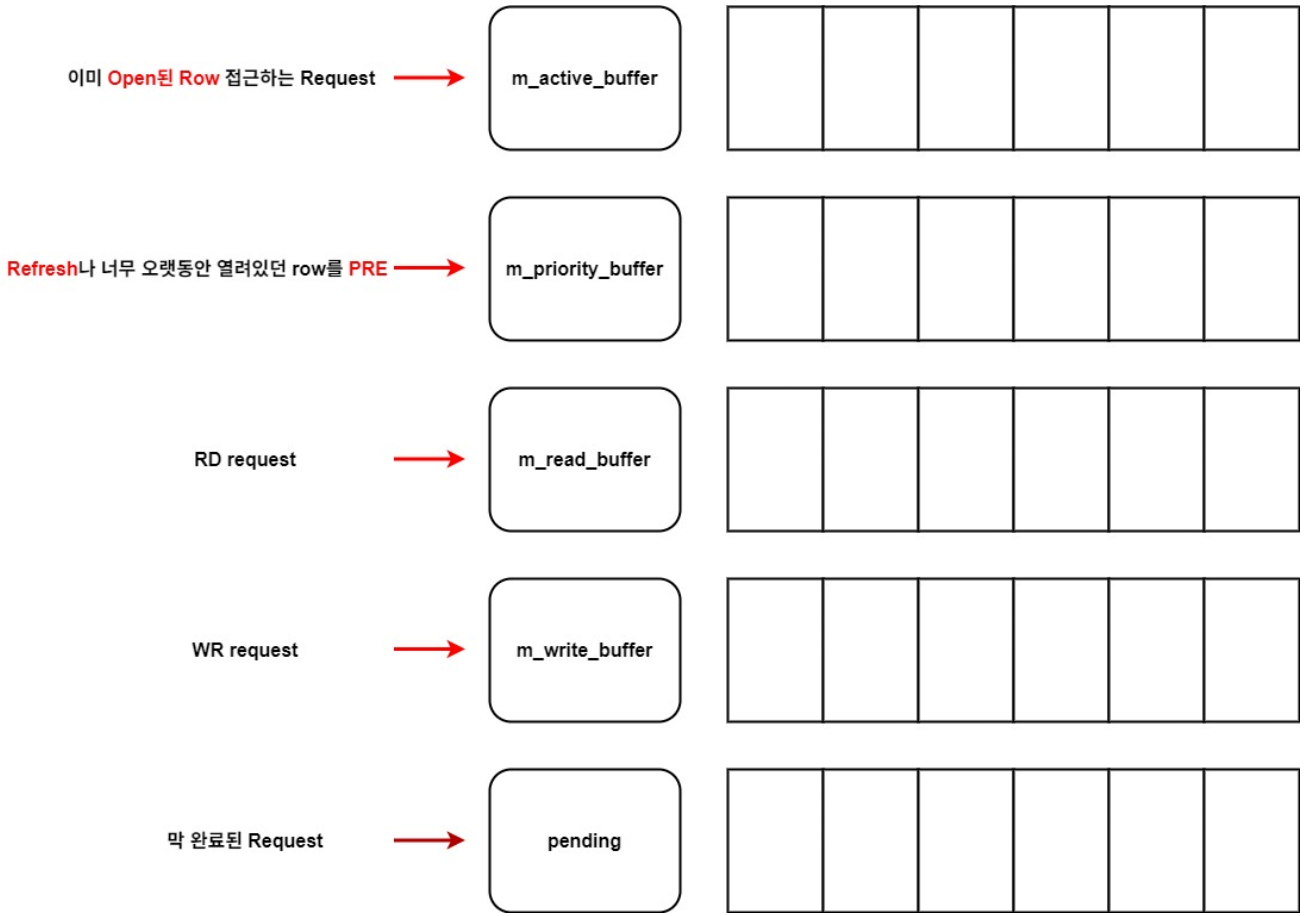
src/memory_system folder

❑ memory_system

- m_memory_system – tick() → m_controllers[channel_id] – tick() → schedule_request(req_it, buffer)

schedule_request()의 Buffers Review

- DRAM Controller에 존재하는 Buffer들



src/memory_system folder

❑ memory_system

- m_memory_system – tick() → m_controllers[channel_id] – tick() → schedule_request(req_it, buffer)

schedule_request()

3.1. 우선 순위가 가장 높은 active buffer를 먼저 확인 (scheduling 기법 – FRFCFS)

3.2. 우선 순위가 다음으로 높은 priority buffer를 확인 (scheduling 기법 – FRFCFS)

- priority 요청은 순서를 깨지 않고 직렬로 처리해야 하는 유지보수성 traffic 이라고 함
- 즉, 3.1과 3.3의 경우 get_best_request() 함수를 통해 buffer를 순회하며 적절한 req를 찾음
- 하지만, 3.2의 경우 priority buffer의 첫번째 req만 확인하고, timing, preq가 안 맞으면 넘어감

3.3. 다음으로, write mode인지 에 따라 read or write buffer를 확인 (scheduling 기법 – FRFCFS)

3.4. active buffer에 있는 대기중인 ACT command 중 지금 발행하려는 req와 같은 bank를 가지는 command가 있으면, request를 취소 함(같은 Bank에는 동시에 한 row만 열려 있음)

src/memory_system folder

❑ memory_system

- m_memory_system – tick() → m_controllers[channel_id] – tick() → schedule_request(req_it, buffer) → get_best_request()

3. schedule_request(req_it, buffer) 예시

1. schedule_request() 호출

2. Scheduler가 buffer에서 best request 선택

3. get_preq_command() 호출 ← Prerequisite 확인 (재귀적으로)
   - ├ Channel 레벨 preq 확인 (등록 없으면 pass)
   - ├ Rank 레벨 preq 확인 (REFab: RequireAllBanksClosed)
   - ├ BankGroup 레벨 preq 확인 (등록 없으면 pass)
   - └ Bank 레벨 preq 확인 (RD/WR: RequireRowOpen)
     - └ Bank 상태가 "Refreshing"이면 ACT 반환 (실행 불가)

4. check_ready(command, addr_vec) 호출 ← Timing Constraints 확인(재귀)
   - ├ Channel 레벨: m_cmd_ready_clk[command] >= clk 확인
   - ├ Rank 레벨: m_cmd_ready_clk[command] >= clk 확인
   - ├ BankGroup 레벨: m_cmd_ready_clk[command] >= clk 확인
   - └ Bank 레벨: m_cmd_ready_clk[command] >= clk 확인
     - └ 모든 레벨 통과 → true 반환

5. request_found = true이면 issue_command() 호출
   - └ update_timing() 호출 → future timing constraints 업데이트

```cpp
ReqBuffer::iterator get_best_request(ReqBuffer& buffer) override {
  if (buffer.size() == 0) {
    return buffer.end();
  }

  for (auto& req : buffer) {
    req.command = m_dram->get_preq_command(req.final_command, req.addr_vec);
  }

  auto candidate = buffer.begin();
  for (auto next = std::next(buffer.begin(), 1); next != buffer.end(); next++) {
    candidate = compare(candidate, next);
  }
  return candidate;
}
};
```

```cpp
bool ready1 = m_dram->check_ready(req1->command, req1->addr_vec);
bool ready2 = m_dram->check_ready(req2->command, req2->addr_vec);

if (ready1 ^ ready2) {
  if (ready1) {
    return req1;
  } else {
    return req2;
  }
}

// Fallback to FCFS
if (req1->arrive <= req2->arrive) {
  return req1;
} else {
  return req2;
}
```

compare()

- Ready가 아니면 Ready인 것
- 둘 다 Ready이면 FCFS

src/memory_system folder

❑ memory_system

- m_memory_system – tick() → m_controllers[channel_id] – tick() → schedule_request(req_it, buffer) → get_best_request() → get_preq_command()

3. schedule_request(req_it, buffer) 예시

1. schedule_request() 호출

2. Scheduler가 buffer에서 best request 선택

3. get_preq_command() 호출 ← Prerequisite 확인 (재귀적으로)
   - ├─ Channel 레벨 preq 확인 (등록 없으면 pass)
   - ├─ Rank 레벨 preq 확인 (REFab: RequireAllBanksClosed)
   - ├─ BankGroup 레벨 preq 확인 (등록 없으면 pass)
   - └─ Bank 레벨 preq 확인 (RD/WR: RequireRowOpen)
     - └─ Bank 상태가 "Refreshing"이면 ACT 반환 (실행 불가)

4. check_ready(command, addr_vec) 호출 ← Timing Constraints 확인(재귀)
   - ├─ Channel 레벨: m_cmd_ready_clk[command] >= clk 확인
   - ├─ Rank 레벨: m_cmd_ready_clk[command] >= clk 확인
   - ├─ BankGroup 레벨: m_cmd_ready_clk[command] >= clk 확인
   - └─ Bank 레벨: m_cmd_ready_clk[command] >= clk 확인
     - └─ 모든 레벨 통과 → true 반환

5. request_found = true이면 issue_command() 호출
   - └─ update_timing() 호출 → future timing constraints 업데이트

```
190   int get_preq_command(int command, const AddrVec_t& addr_vec, Clk_t m_clk) {
191     int child_id = addr_vec[m_level + 1];
192     if (m_spec->m_preqs[m_level][command]) {
193       int preq_cmd = m_spec->m_preqs[m_level][command](static_cast<NodeType*>(this), command, addr_vec, m_clk);
194       if (preq_cmd != -1) {
195         // stop recursion: there is a prerequisite at this level
196         return preq_cmd;
197       }
198     }
199
200     if (!m_child_nodes.size()) {
201       // stop recursion: there were no prequisites at any level
202       return command;
203     }
204
205     // recursively get preq command at my child
206     return m_child_nodes[child_id]->get_preq_command(command, addr_vec, m_clk);
207   };
```

재귀

```
511   void set_preqs() {
512     m_preqs.resize(m_levels.size(), std::vector<PreqFunc_t<Node>>(m_commands.size()));
513
514     // Rank Actions
515     m_preqs[m_levels["rank"]][m_commands["REFab"]] = Lambdas::Preq::Rank::RequireAllBanksClosed<DDR4>;
516
517     // Bank actions
518     m_preqs[m_levels["bank"]][m_commands["RD"]] = Lambdas::Preq::Bank::RequireRowOpen<DDR4>;
519     m_preqs[m_levels["bank"]][m_commands["WR"]] = Lambdas::Preq::Bank::RequireRowOpen<DDR4>;
520     m_preqs[m_levels["bank"]][m_commands["ACT"]] = Lambdas::Preq::Bank::RequireRowOpen<DDR4>;
521     m_preqs[m_levels["bank"]][m_commands["PRE"]] = Lambdas::Preq::Bank::RequireBankClosed<DDR4>;
522   };
```

src/memory_system folder

❑ memory_system

- m_memory_system – tick() → m_controllers[channel_id] – tick() → schedule_request(req_it, buffer) → check_ready()

3. schedule_request(req_it, buffer) 예시

1. schedule_request() 호출

2. Scheduler가 buffer에서 best request 선택

3. get_preq_command() 호출 ← Prerequisite 확인 (재귀적으로)
   - ├─ Channel 레벨 preq 확인 (등록 없으면 pass)
   - ├─ Rank 레벨 preq 확인 (REFab: RequireAllBanksClosed)
   - ├─ BankGroup 레벨 preq 확인 (등록 없으면 pass)
   - └─ Bank 레벨 preq 확인 (RD/WR: RequireRowOpen)
     - └─ Bank 상태가 "Refreshing"이면 ACT 반환 (실행 불가)

4. check_ready(command, addr_vec) 호출 ← Timing Constraints 확인(재귀)
   - ├─ Channel 레벨: m_cmd_ready_clk[command] >= clk 확인
   - ├─ Rank 레벨: m_cmd_ready_clk[command] >= clk 확인
   - ├─ BankGroup 레벨: m_cmd_ready_clk[command] >= clk 확인
   - └─ Bank 레벨: m_cmd_ready_clk[command] >= clk 확인
     - └─ 모든 레벨 통과 → true 반환

5. request_found = true이면 issue_command() 호출
   - └─ update_timing() 호출 → future timing constraints 업데이트

```
250     bool check_ready(int command, const AddrVec_t& addr_vec) override {
251       int channel_id = addr_vec[m_levels["channel"]];
252       return m_channels[channel_id]->check_ready(command, addr_vec, m_clk);
253     };
```

```
209     bool check_ready(int command, const AddrVec_t& addr_vec, Clk_t clk) {
210       if (m_cmd_ready_clk[command] != -1 && clk < m_cmd_ready_clk[command]) {
211         // stop recursion: the check failed at this level
212         return false;
213       }
214
215       int child_id = addr_vec[m_level+1];
216       if (m_level == m_spec->m_command_scopes[command] || !m_child_nodes.size()) {
217         // stop recursion: the check passed at all levels
218         return true;
219       }
220
221       if (child_id == -1) {
222         // if it is a same bank command, recurse all children in rank level
223         bool ready = true;
224         for (auto child : m_child_nodes) {
225           ready = ready && child->check_ready(command, addr_vec, clk);
226         }
227         return ready;
228       } else {
229         // recursively check my child
230         return m_child_nodes[child_id]->check_ready(command, addr_vec, clk);
231       }
232     };
```

재귀

src/memory_system folder

❑ memory_system

- m_memory_system – tick() → m_controllers[channel_id] – tick()

3. schedule_request(req_it, buffer) 예시

1. schedule_request() 호출

2. Scheduler가 buffer에서 best request 선택

3. get_preq_command() 호출 ← Prerequisite 확인 (재귀적으로)
   - ├─ Channel 레벨 preq 확인 (등록 없으면 pass)
   - ├─ Rank 레벨 preq 확인 (REFab: RequireAllBanksClosed)
   - ├─ BankGroup 레벨 preq 확인 (등록 없으면 pass)
   - └─ Bank 레벨 preq 확인 (RD/WR: RequireRowOpen)
     - └─ Bank 상태가 "Refreshing"이면 ACT 반환 (실행 불가)

4. check_ready(command, addr_vec) 호출 ← Timing Constraints 확인(재귀)
   - ├─ Channel 레벨: m_cmd_ready_clk[command] >= clk 확인
   - ├─ Rank 레벨: m_cmd_ready_clk[command] >= clk 확인
   - ├─ BankGroup 레벨: m_cmd_ready_clk[command] >= clk 확인
   - └─ Bank 레벨: m_cmd_ready_clk[command] >= clk 확인
     - └─ 모든 레벨 통과 → true 반환

5. schedule_request의 return값: request_found = true이면 issue_command() 호출
   - └─ update_timing() 호출 → future timing constraints 업데이트

```
201     // 4. Finally, issue the commands to serve the request
202     if (request_found) {
203       // If we find a real request to serve
204       if (req_it->is_stat_updated == false) {
205         update_request_stats(req_it); // 히트/미스 통계
206       }
207       m_dram->issue_command(req_it->command, req_it->addr_vec)
208
```

```
210     void issue_command(int command, const AddrVec_t& addr_vec) override {
211       int channel_id = addr_vec[m_levels["channel"]];
212       m_channels[channel_id]->update_timing(command, addr_vec, m_clk);
213       m_channels[channel_id]->update_powers(command, addr_vec, m_clk);
214       m_channels[channel_id]->update_states(command, addr_vec, m_clk);
215
216       // Check if the command requires future action
217       check_future_action(command, addr_vec);
218     };
```

src/memory_system folder

❑ memory_system

- m_memory_system – tick() → m_controllers[channel_id] – tick()

```cpp
void tick() override {
  m_clk++;

  // Update statistics
  s_queue_len += m_read_buffer.size() + m_write_buffer.size() + pending.size();
  s_read_queue_len += m_read_buffer.size() + pending.size();
  s_write_queue_len += m_write_buffer.size();
  s_priority_queue_len += m_priority_buffer.size();

  // 1. Serve completed reads - 완료된 읽기 요청 처리
  serve_completed_reads();

  m_refresh->tick();  // 리프레시 관리

  // 2. Try to find a request to serve.- 스케줄링 (어떤 요청을
  ReqBuffer::iterator req_it;
  ReqBuffer* buffer = nullptr;
  bool request_found = schedule_request(req_it, buffer);

  // 2.1 Take row policy action
  m_rowpolicy->update(request_found, req_it);

  // 3. Update all plugins
  for (auto plugin : m_plugins) {
    plugin->update(request_found, req_it);
  }
```

```cpp
void update(bool request_found, ReqBuffer::iterator& req_it) override {

  if (!request_found)
    return;

  if (m_dram->m_command_meta(req_it->command).is_closing ||
      m_dram->m_command_meta(req_it->command).is_refreshing)  // PRE or REF
  {
    // All Bank Closes (PREA, REFab, PRE) - 서로 다른 명령어에 의해 case가 나뉨
    if (req_it->addr_vec[m_bankgroup_level] == -1 && req_it->addr_vec[m_bank_level] == -1) {  // all bank closes
      // 모든 Bank의 카운터를 0으로 리셋
      for (int b = 0; b < m_num_banks; b++) {
        for (int bg = 0; bg < m_num_bankgroups; bg++) {
          int rank_id = req_it->addr_vec[m_rank_level];
          int flat_bank_id = b + bg * m_num_banks + rank_id * m_num_banks * m_num_bankgroups;
          m_col_accesses[flat_bank_id] = 0;
        }
      }
    } else if (req_it->addr_vec[m_bankgroup_level] == -1) {  // same bank closes
      // 동일한 Bank 번호를 가진 모든 BankGroup의 카운터 리셋
      for (int bg = 0; bg < m_num_bankgroups; bg++) {
        int bank_id = req_it->addr_vec[m_bank_level];
        int rank_id = req_it->addr_vec[m_rank_level];
        int flat_bank_id = bank_id + bg * m_num_banks + rank_id * m_num_banks * m_num_bankgroups;
        m_col_accesses[flat_bank_id] = 0;
      }
    } else {  // single bank closes  (PRE, VRR, RDA, WRA) - Read + Auto-Precharge / Write + Auto-Precharge
      // 단일 Bank만 닫음
      int flat_bank_id = req_it->addr_vec[m_bank_level] +
                         req_it->addr_vec[m_bankgroup_level] * m_num_banks +
                         req_it->addr_vec[m_rank_level] * m_num_banks * m_num_bankgroups;

      m_col_accesses[flat_bank_id] = 0;
    }
  } else if (m_dram->m_command_meta(req_it->command).is_accessing)  // RD or WR - Accessing 명령 (RD, WR)
  {
    int flat_bank_id = req_it->addr_vec[m_bank_level] +
                       req_it->addr_vec[m_bankgroup_level] * m_num_banks +
                       req_it->addr_vec[m_rank_level] * m_num_banks * m_num_bankgroups;

    m_col_accesses[flat_bank_id]++; // 카운터 증가

    if (m_col_accesses[flat_bank_id] >= m_cap) {
      // Cap 초과 → 강제로 PRE 요청 발행
      Request req(req_it->addr_vec, m_PRE_req_id);
      m_ctrl->priority_send(req);
      m_col_accesses[flat_bank_id] = 0;
      s_num_close_reqs++;
    }
  }
};
```

basic_rowpolicies.cpp -> update()

```cpp
            if (m_active_buffer.enqueue(req_it)) {
              buffer->remove(req_it);
            }
          }
        }
      }

    }

};
```

4. Row Policy를 적용 (update) → 특정 column이 너무 오랫동안 open되어 있으면 PRE(close명령)을 priority queue에 넣음

5. Plugin을 적용 (update)

src/memory_system folder

❑ memory_system

- m_memory_system – tick() → m_controllers[channel_id] – tick()

```cpp
void tick() override {
  m_clk++;

  // Update statistics
  s_queue_len += m_read_buffer.size() + m_write_buffer.size() + m_priority_buffer.size() + pending.size();
  s_read_queue_len += m_read_buffer.size() + pending.size();
  s_write_queue_len += m_write_buffer.size();
  s_priority_queue_len += m_priority_buffer.size();

  // 1. Serve completed reads - 완료된 읽기 요청 처리
  serve_completed_reads();

  m_refresh->tick();  // 리프레시 관리

  // 2. Try to find a request to serve.- 스케줄링 (어떤 요청을 발행할지 결정)
  ReqBuffer::iterator req_it;
  ReqBuffer* buffer = nullptr;
  bool request_found = schedule_request(req_it, buffer);

  // 2.1 Take row policy action
  m_rowpolicy->update(request_found, req_it);

  // 3. Update all plugins
  for (auto plugin : m_plugins) {
    plugin->update(request_found, req_it);
  }

  // 4. Finally, issue the commands to serve the request
  if (request_found) {
    // If we find a real request to serve
    if (req_it->is_stat_updated == false) {
      update_request_stats(req_it); // 히트/미스 통계
    }
    m_dram->issue_command(req_it->command, req_it->addr_vec);

    // If we are issuing the last command, set depart clock cycle and move the request to the pending queue
    if (req_it->command == req_it->final_command) {
      if (req_it->type_id == Request::Type::Read) {
        req_it->depart = m_clk + m_dram->m_read_latency;
        pending.push_back(*req_it);
      } else if (req_it->type_id == Request::Type::Write) {
        // TODO: Add code to update statistics
      }
      buffer->remove(req_it);
    } else {
      if (m_dram->m_command_meta(req_it->command).is_opening) {
        if (m_active_buffer.enqueue(*req_it)) {
          buffer->remove(req_it);
        }
      }
    }
  }
};
```

```cpp
void update_request_stats(ReqBuffer::iterator& req)
{
  req->is_stat_updated = true;

  if (req->type_id == Request::Type::Read)
  {
    if (is_row_hit(req)) {
      s_read_row_hits++;
      s_row_hits++;
      if (req->source_id != -1)
        s_read_row_hits_per_core[req->source_id]++;
    } else if (is_row_open(req)) {
      s_read_row_conflicts++;
      s_row_conflicts++;
      if (req->source_id != -1)
        s_read_row_conflicts_per_core[req->source_id]++;
    } else {
      s_read_row_misses++;
      s_row_misses++;
      if (req->source_id != -1)
        s_read_row_misses_per_core[req->source_id]++;
    }
  }
  else if (req->type_id == Request::Type::Write)
  {
    if (is_row_hit(req)) {
      s_write_row_hits++;
      s_row_hits++;
    } else if (is_row_open(req)) {
      s_write_row_conflicts++;
      s_row_conflicts++;
    } else {
      s_write_row_misses++;
      s_row_misses++;
    }
  }
}
```

```cpp
inline static const ImplLUT m_command_meta = LUT<DRAMCommandMeta> (
  m_commands, {
                          // open?   close?   access?  refresh?
    {"ACT",       {true,   false,   false,   false}},
    {"PRE",       {false,  true,    false,   false}},
    {"PREA",      {false,  true,    false,   false}},
    {"RD",        {false,  false,   true,    false}},
    {"WR",        {false,  false,   true,    false}},
    {"RDA",       {false,  true,    true,    false}},
    {"WRA",       {false,  true,    true,    false}},
    {"REFab",     {false,  false,   false,   true }},
    {"REFab_end", {false,  true,    false,   false}},
  }
);
```

6. Statistics를 update함

7. Scheduling이 완료된 Request를 issue하고

8.1. 해당 Request의 current command가 last command이면, pending buffer에 넣음

8.2. 해당 Request의 current command가 last command가 아니고, row를 open하면, active buffer에 넣음

src/memory_system folder

❑ memory_system

- m_memory_system – tick() → m_controllers[channel_id] – tick()

```cpp
void tick() override {
  m_clk++;

  // Update statistics
  s_queue_len += m_read_buffer.size() + m_write_buffer.size() + m_priority_buffer.size() + pending.size();
  s_read_queue_len += m_read_buffer.size() + pending.size();
  s_write_queue_len += m_write_buffer.size();
  s_priority_queue_len += m_priority_buffer.size();

  // 1. Serve completed reads - 완료된 읽기 요청 처리
  serve_completed_reads();

  m_refresh->tick();  // 리프레시 관리

  // 2. Try to find a request to serve.- 스케줄링 (어떤 요청을 발행할지 결정)
  ReqBuffer::iterator req_it;
  ReqBuffer* buffer = nullptr;
  bool request_found = schedule_request(req_it, buffer);

  // 2.1 Take row policy action
  m_rowpolicy->update(request_found, req_it);

  // 3. Update all plugins
  for (auto plugin : m_plugins) {
    plugin->update(request_found, req_it);
  }
```

```cpp
void issue_command(int command, const AddrVec_t& addr_vec) override {
  int channel_id = addr_vec[m_levels["channel"]];
  m_channels[channel_id]->update_timing(command, addr_vec, m_clk);
  m_channels[channel_id]->update_powers(command, addr_vec, m_clk);
  m_channels[channel_id]->update_states(command, addr_vec, m_clk);

  // Check if the command requires future action
  check_future_action(command, addr_vec);
};
```

```cpp
// 4. Finally, issue the commands to serve the request
if (request_found) {
  // If we find a real request to serve
  if (req_it->is_stat_updated == false) {
    update_request_stats(req_it); // 히트/미스 통계
  }
  m_dram->issue_command(req_it->command, req_it->addr_vec);

  // If we are issuing the last command, set depart clock cycle and move the request to the pending queue
  if (req_it->command == req_it->final_command) {
    if (req_it->type_id == Request::Type::Read) {
      req_it->depart = m_clk + m_dram->m_read_latency;
      pending.push_back(*req_it);
    } else if (req_it->type_id == Request::Type::Write) {
      // TODO: Add code to update statistics
    }
    buffer->remove(req_it);
  } else {
    if (m_dram->m_command_meta(req_it->command).is_opening) {
      if (m_active_buffer.enqueue(*req_it)) {
        buffer->remove(req_it);
      }
    }
  }
}
};
```

```cpp
inline static const ImplLUT m_command_scopes = LUT (
  m_commands, m_levels, {
    {"ACT",   "row"},
    {"PRE",   "bank"},   {"PREA",   "rank"},
    {"RD",    "column"}, {"WR",     "column"}, {"RDA",   "column"}, {"WRA",
    "column"},
    {"REFab", "rank"},   {"REFab_end", "rank"},
  }
);
```

- issue_command 동작
  - 해당 명령어를 기준으로 timing constraints를 갱신 (ex. ACT command의 경우, nRCD 후 RD/WR 가능)
  - 해당 명령어를 기준으로 command scope과 해당하는 rank별로 power counter 갱신
  - 해당 명령어를 기준으로 command scope과 state 갱신 (ex. ACT의 경우, scope = row, stated = opened)

src/memory_system folder

❑ memory_system

- m_memory_system – tick() → m_controllers[channel_id] – tick() → m_dram->issue_command(req_it->command, req_it->addr_vec)

```
210     void issue_command(int command, const AddrVec_t& addr_vec) override {
211       int channel_id = addr_vec[m_levels["channel"]];
212       m_channels[channel_id]->update_timing(command, addr_vec, m_clk);
213       m_channels[channel_id]->update_powers(command, addr_vec, m_clk);
214       m_channels[channel_id]->update_states(command, addr_vec, m_clk);
215
216       // Check if the command requires future action
217       check_future_action(command, addr_vec);
218     };
```

```
84      inline static const ImplLUT m_command_scopes = LUT (
85        m_commands, m_levels, {
86          {"ACT",    "row"},
87          {"PRE",    "bank"},   {"PREA",    "rank"},
88          {"RD",     "column"}, {"WR",      "column"}, {"RDA",    "column"}, {"WRA",
            "column"},
89          {"REFab", "rank"},  {"REFab_end", "rank"},
90        }
91      );
```

- issue_command 동작
  - 해당 명령어를 기준으로 timing constraints를 갱신 (ex. ACT command의 경우, nRCD 후 RD/WR 가능)
  - 해당 명령어를 기준으로 command scope과 해당하는 rank별로 power counter 갱신
  - 해당 명령어를 기준으로 command scope과 state 갱신 (ex. ACT의 경우, scope = row, stated = opened)

src/memory_system folder

❑ memory_system

- m_memory_system – tick()

- abcd

src/memory_system folder

❑ memory_system

- m_memory_system – tick()

```
93   inline static const ImplLUT m_command_meta = LUT<DRAMCommandMeta> (
94     m_commands, {
95                                 // open?   close?   access?  refresh?
96       {"ACT",        {true,   false,   false,   false}},
97       {"PRE",        {false,  true,    false,   false}},
98       {"PREA",       {false,  true,    false,   false}},
99       {"RD",         {false,  false,   true,    false}},
100      {"WR",         {false,  false,   true,    false}},
101      {"RDA",        {false,  true,    true,    false}},
102      {"WRA",        {false,  true,    true,    false}},
103      {"REFab",      {false,  false,   false,   true }},
104      {"REFab_end",  {false,  true,    false,   false}},
105    }
106  );
```

- abcd

src/memory_system folder

❑ memory_system

- m_memory_system – tick()

```
93  inline static const ImplLUT m_command_meta = LUT<DRAMCommandMeta> (
94    m_commands, {
95                          // open?   close?   access?  refresh?
96      {"ACT",       {true,   false,   false,   false}},
97      {"PRE",       {false,  true,    false,   false}},
98      {"PREA",      {false,  true,    false,   false}},
99      {"RD",        {false,  false,   true,    false}},
100     {"WR",        {false,  false,   true,    false}},
101     {"RDA",       {false,  true,    true,    false}},
102     {"WRA",       {false,  true,    true,    false}},
103     {"REFab",     {false,  false,   false,   true }},
104     {"REFab_end", {false,  true,    false,   false}},
105   }
106 );
```

- abcd

src/addr_mapper folder

- addr_mapper
  - abcd

src/addr_mapper folder

- addr_mapper
  - abcd

src/translation folder

- translation

  - abcd

src/translation folder

- translation

  - abcd

src/translation folder

- translation

  - abcd

src/dram folder

- dram
  - abcd

src/dram folder

- dram
  - abcd

src/dram folder

- dram
  - abcd

src/dram_controller folder

❑ DRAM Controller Operations

- **class GenericDRAMController – init()**

(src\dram_controller\impl\ generic_dram_controller.cpp)

```
54      void init() override {
55        m_wr_low_watermark =  param<float>("wr_low_watermark").desc("Threshold for switching back to read mode.").default_val(0.2f)
56        m_wr_high_watermark = param<float>("wr_high_watermark").desc("Threshold for switching to write mode.").default_val(0.8f);
57
58        m_scheduler = create_child_ifce<IScheduler>();
59        m_refresh = create_child_ifce<IRefreshManager>();
60        m_rowpolicy = create_child_ifce<IRowPolicy>();
61
62        if (m_config["plugins"]) {
63          YAML::Node plugin_configs = m_config["plugins"];
64          for (YAML::iterator it = plugin_configs.begin(); it != plugin_configs.end(); ++it) {
65            m_plugins.push_back(create_child_ifce<IControllerPlugin>(*it));
66          }
67        }
68      };
```

1.1. YAML file을 통해 write buffer priority threshold를 초기화.
→ Scheduler가 사용할 buffer를 결정
1.2. Scheduler, Refresh Manager, Row Policy를 생성.
1.3. Plugin을 load하여 Controller를 확장

```
// 2.2.1    If no request to be scheduled in the priority buffer, check the read and write buffers.
if (!request_found) {
  // Query the write policy to decide which buffer to serve
  set_write_mode();
  auto& buffer = m_is_write_mode ? m_write_buffer : m_read_buffer;
  if (req_it = m_scheduler->get_best_request(buffer); req_it != buffer.end()) {
    request_found = m_dram->check_ready(req_it->command, req_it->addr_vec);
    req_buffer = &buffer;
  }
}
```

src/dram_controller folder

❑ DRAM Controller Operations

- **class GenericDRAMController – setup()**

```
70  void setup(IFrontEnd* frontend, IMemorySystem* memory_system) override {
71    m_dram = memory_system->get_ifce<IDRAM>();
72    m_bank_addr_idx = m_dram->m_levels("bank");
73    m_priority_buffer.max_size = 512*3 + 32;
74
75    m_num_cores = frontend->get_num_cores();
76
77    s_read_row_hits_per_core.resize(m_num_cores, 0);
78    s_read_row_misses_per_core.resize(m_num_cores, 0);
79    s_read_row_conflicts_per_core.resize(m_num_cores, 0);
80
81    register_stat(s_row_hits).name("row_hits_{}", m_channel_id);
82    register_stat(s_row_misses).name("row_misses_{}", m_channel_id);
83    register_stat(s_row_conflicts).name("row_conflicts_{}", m_channel_id);
84    register_stat(s_read_row_hits).name("read_row_hits_{}", m_channel_id);
85    register_stat(s_read_row_misses).name("read_row_misses_{}", m_channel_id);
86    register_stat(s_read_row_conflicts).name("read_row_conflicts_{}", m_channel_id);
87    register_stat(s_write_row_hits).name("write_row_hits_{}", m_channel_id);
88    register_stat(s_write_row_misses).name("write_row_misses_{}", m_channel_id);
89    register_stat(s_write_row_conflicts).name("write_row_conflicts_{}", m_channel_id);
90
91    for (size_t core_id = 0; core_id < m_num_cores; core_id++) {
92      register_stat(s_read_row_hits_per_core[core_id]).name("read_row_hits_core_{}", core_id);
93      register_stat(s_read_row_misses_per_core[core_id]).name("read_row_misses_core_{}", core_id);
94      register_stat(s_read_row_conflicts_per_core[core_id]).name("read_row_conflicts_core_{}", core_id);
95    }
96
97    register_stat(s_num_read_reqs).name("num_read_reqs_{}", m_channel_id);
98    register_stat(s_num_write_reqs).name("num_write_reqs_{}", m_channel_id);
99    register_stat(s_num_other_reqs).name("num_other_reqs_{}", m_channel_id);
100   register_stat(s_queue_len).name("queue_len_{}", m_channel_id);
101   register_stat(s_read_queue_len).name("read_queue_len_{}", m_channel_id);
102   register_stat(s_write_queue_len).name("write_queue_len_{}", m_channel_id);
103   register_stat(s_priority_queue_len).name("priority_queue_len_{}", m_channel_id);
104   register_stat(s_queue_len_avg).name("queue_len_avg_{}", m_channel_id);
105   register_stat(s_read_queue_len_avg).name("read_queue_len_avg_{}", m_channel_id);
106   register_stat(s_write_queue_len_avg).name("write_queue_len_avg_{}", m_channel_id);
107   register_stat(s_priority_queue_len_avg).name("priority_queue_len_avg_{}", m_channel_id);
108
109   register_stat(s_read_latency).name("read_latency_{}", m_channel_id);
110   register_stat(s_avg_read_latency).name("avg_read_latency_{}", m_channel_id);
111 };
```

2.1. DRAM Interface 연결 - (ex. dram.h → DDR4)

2.2. Bank level의 index를 조회

- DRAM Address Vector → ex. addr_vec[0] = channel, [1] = rank, [2] = bankgroup, [3] = bank, [4] = row, [5] = column
- 이 경우, Bank level index = 3
- 아래는 Row Hit 판별을 위해 Bank level index가 어떻게 사용되는지를 나타냄
- m_bank_addr_idx = 3 → bank = addr_vec[3] = 2 → Row Hit Check: open_rows[bank=2]의 row=1024 check → hit?

2.3. priority buffer Size를 하드 코딩

src/dram_controller folder

❑ DRAM Controller Operations

- **class GenericDRAMController – setup()**

```
70  void setup(IFrontEnd* frontend, IMemorySystem* memory_system) override {
71    m_dram = memory_system->get_ifce<IDRAM>();
72    m_bank_addr_idx = m_dram->m_levels("bank");
73    m_priority_buffer.max_size = 512*3 + 32;
74
75    m_num_cores = frontend->get_num_cores();
76
77    s_read_row_hits_per_core.resize(m_num_cores, 0);
78    s_read_row_misses_per_core.resize(m_num_cores, 0);
79    s_read_row_conflicts_per_core.resize(m_num_cores, 0);
80
81    register_stat(s_row_hits).name("row_hits_{}", m_channel_id);
82    register_stat(s_row_misses).name("row_misses_{}", m_channel_id);
83    register_stat(s_row_conflicts).name("row_conflicts_{}", m_channel_id);
84    register_stat(s_read_row_hits).name("read_row_hits_{}", m_channel_id);
85    register_stat(s_read_row_misses).name("read_row_misses_{}", m_channel_id);
86    register_stat(s_read_row_conflicts).name("read_row_conflicts_{}", m_channel_id);
87    register_stat(s_write_row_hits).name("write_row_hits_{}", m_channel_id);
88    register_stat(s_write_row_misses).name("write_row_misses_{}", m_channel_id);
89    register_stat(s_write_row_conflicts).name("write_row_conflicts_{}", m_channel_id);
90
91    for (size_t core_id = 0; core_id < m_num_cores; core_id++) {
92      register_stat(s_read_row_hits_per_core[core_id]).name("read_row_hits_core_{}", core_id);
93      register_stat(s_read_row_misses_per_core[core_id]).name("read_row_misses_core_{}", core_id);
94      register_stat(s_read_row_conflicts_per_core[core_id]).name("read_row_conflicts_core_{}", core_id);
95    }
96
97    register_stat(s_num_read_reqs).name("num_read_reqs_{}", m_channel_id);
98    register_stat(s_num_write_reqs).name("num_write_reqs_{}", m_channel_id);
99    register_stat(s_num_other_reqs).name("num_other_reqs_{}", m_channel_id);
100   register_stat(s_queue_len).name("queue_len_{}", m_channel_id);
101   register_stat(s_read_queue_len).name("read_queue_len_{}", m_channel_id);
102   register_stat(s_write_queue_len).name("write_queue_len_{}", m_channel_id);
103   register_stat(s_priority_queue_len).name("priority_queue_len_{}", m_channel_id);
104   register_stat(s_queue_len_avg).name("queue_len_avg_{}", m_channel_id);
105   register_stat(s_read_queue_len_avg).name("read_queue_len_avg_{}", m_channel_id);
106   register_stat(s_write_queue_len_avg).name("write_queue_len_avg_{}", m_channel_id);
107   register_stat(s_priority_queue_len_avg).name("priority_queue_len_avg_{}", m_channel_id);
108
109   register_stat(s_read_latency).name("read_latency_{}", m_channel_id);
110   register_stat(s_avg_read_latency).name("avg_read_latency_{}", m_channel_id);
111 };
```

2.4. Frontend에서 core 수를 조회하여 m_num_cores에 저장

2.5. Read Row Hit Per Core, … , Read Row Conflict Per Core 등의 배열 초기화 → multi core 지원을 위해 초기화

→ 위 배열들은 통계를 수집하고 출력 용도로만 사용됨

2.6. 다양한 통계 variable를 register하여 DRAM controller의 동작을 분석 → 통계를 수집하고 출력

src/dram_controller folder

❑ DRAM Controller Operations

- class GenericDRAMController – send()

```
113   bool send(Request& req) override {
114     req.final_command = m_dram->m_request_translations(req.type_id);
115
116     switch (req.type_id) {
117       case Request::Type::Read: {
118         s_num_read_reqs++;
119         break;
120       }
121       case Request::Type::Write: {
122         s_num_write_reqs++;
123         break;
124       }
125       default: {
126         s_num_other_reqs++;
127         break;
128       }
129     }
130
131     // Forward existing write requests to incoming read requests
132     if (req.type_id == Request::Type::Read) {
133       auto compare_addr = [req](const Request& wreq) {
134         return wreq.addr == req.addr;
135       };
136       if (std::find_if(m_write_buffer.begin(), m_write_buffer.end(), compare_addr) != m_write_buffer.end()) {
137         // The request will depart at the next cycle
138         req.depart = m_clk + 1;
139         pending.push_back(req);
140         return true;
141       }
142     }
143
144     // Else, enqueue them to corresponding buffer based on request type id
145     bool is_success = false;
146     req.arrive = m_clk;
147     if        (req.type_id == Request::Type::Read) {
148       is_success = m_read_buffer.enqueue(req);
149     } else if (req.type_id == Request::Type::Write) {
150       is_success = m_write_buffer.enqueue(req);
151     } else {
152       throw std::runtime_error("Invalid request type!");
153     }
154     if (!is_success) {
155       // We could not enqueue the request
156       req.arrive = -1;
157       return false;
158     }
159
160     return true;
161   };
```

src/dram_controller folder

- dram_controller

  - abcd

src/base folder

- base
  - abcd