# 컴퓨터 공학 기초 실험2 보고서

## 실험제목: FIFO

실험일자: 2018년 10월 10일 (수)

제출일자: 2018년 10월 23일 (화)


학 과: 컴퓨터공학과

담당교수: 공진흥 교수님

실습분반: 수요일 0, 1, 2

학 번: 2015722025

성 명: 정 용 훈

# 1. 제목 및 목적

## A. 제목

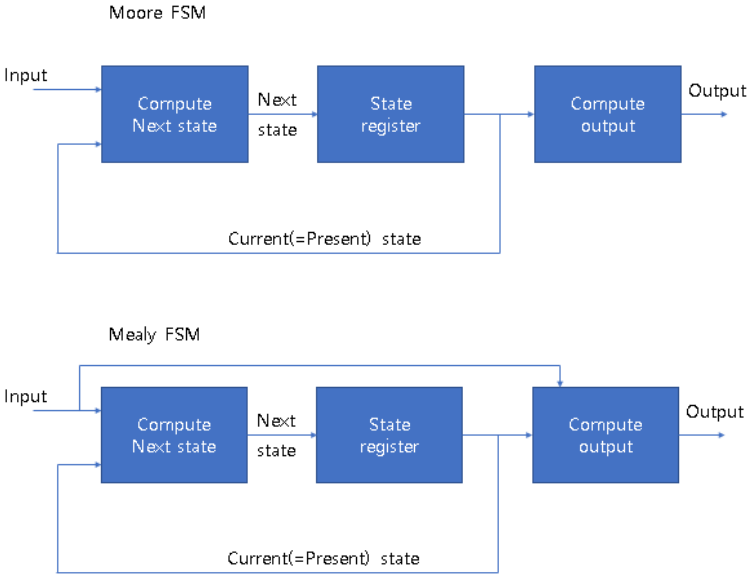
FIFO

## B. 목적

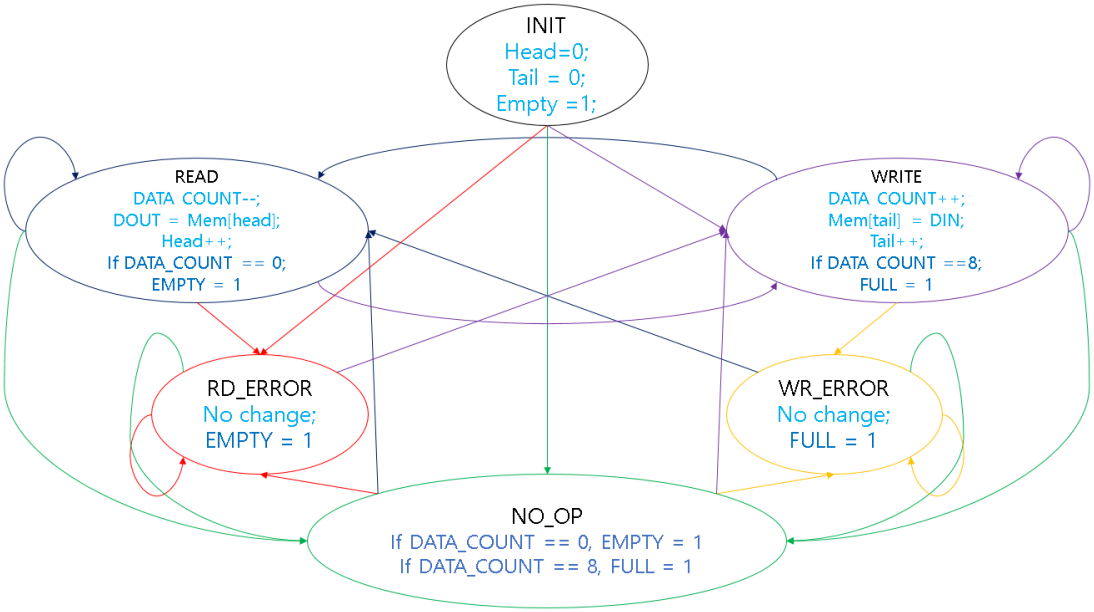
FIFO자료구조에서 정보가 저장되고 정보가 삭제되는 구조를 이해하고 이를 구현한다. FIFO의 구조를 확인하기 위하여 정보가 저장되고 출력하는 상태를 FSM으로 나타내며 전에 구현한 Register File을 통하여 wave form을 통해 가시적으로 결과를 확인한다.

## 2. 원리(배경지식)

해당 실습인 FIFO를 수행하기 위해서는 FSM에 대한 개념과 전 실습에 구현한 Register File, 마지막으로 First In First Out에 관련한 개념을 이해하고 있어야 합니다. FSM이란 현 상태와 Input에 의해 다음 상태가 결정되며 결정된 상태에 따라 Output의 값을 결정하게 됩니다. Output의 값이 결정되는 조건에 따라 Moore와 FSM인지 Mealy FSM인지 결정이 됩니다. Moore FSM은 Output의 값이 현상태에만 의존하지만 Mealy의 Output값은 현상태와 Input에 의해 결정이 된다는 차이점이 있습니다. 아래 이미지는 두 종류의 FSM이 작동하는 방식을 그림으로 나타낸 것입니다.

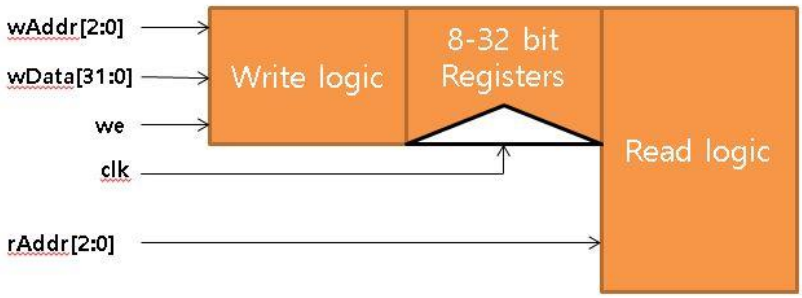



이번 FIFO 실습에서는 아래와 같은 그림으로 FSM이 작동합니다.

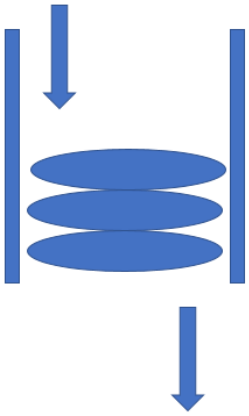



해당 FSM은 Input에 관련하지 않고 현 상태에 따라 동작이 결정되어 Output이 결정되므로 Moore FSM을 기반으로 하는 동작입니다

다음으로는 Register File에 관련한 개념과 이해입니다. Register File의 동작 방법으로는 write enable의 값을 신호로 정보를 저장할 주소와 정보를 Input으로 받아 Write logic에서 해당 주소에 정보를 저장합니다. 이를 기반으로 Read logic에서는 주소의 값을 Input으로 받으며 해당 주소에 저장되어 있는 데이터의 값을 불러오는 동작을 합니다. 아래 이미지는 구현된 Register File을 간단하게 나타낸 이미지입니다.




다음으로는 해당 실습에 필요한 마지막 개념인 FIFO(First In First Out)입니다. FIFO란 자료구조의 일종으로 말그대로 먼저 들어온 정보는 먼저 삭제되는 규칙을 가지고 있는 자료구조입니다. 아래 그림처럼 먼저 들어간 원은 삭제될 때 가장 먼저 나가는 것을 확인 할 수 있습니다.



위 개념을 종합적으로 생각했을 때 이번 설계를 하게 될 FIFO는 FSM을 통해 모듈의 동작과 Output이 결정되며 해당 동작에는 Register File이 연관 되어있어 정보를 저장하고 출력하는 것을 수행하게 됩니다. 이 과정에서 FIFO의 규칙을 따라 먼저 저장된 정보는 wave form을 통해 가장 먼저 출력되는 것을 확인 할 수 있으며 정보의 상태가 Full이면 Full에 대한 Output과 Empty이면 Empty에 대한 Output이 구분되 출력되는 것을 확인 할 수 있습니다.

## 3. 설계 세부사항

(1) Module configuration

| 구분 | 이름 | 설명 |
|---|---|---|
| Top module | fifo | FIFO의 top module |
| Sub module | fifo_ns | Next state module(top module에서 instance) |
| Sub module | fifo_cal | Calculate address module(top module에서 instance) |
| Sub module | fifo_out | Output logic module(top module에서 instance) |
| Sub module | Register_file | Register file module(top module에서 instance) |

(2) I/O configuration

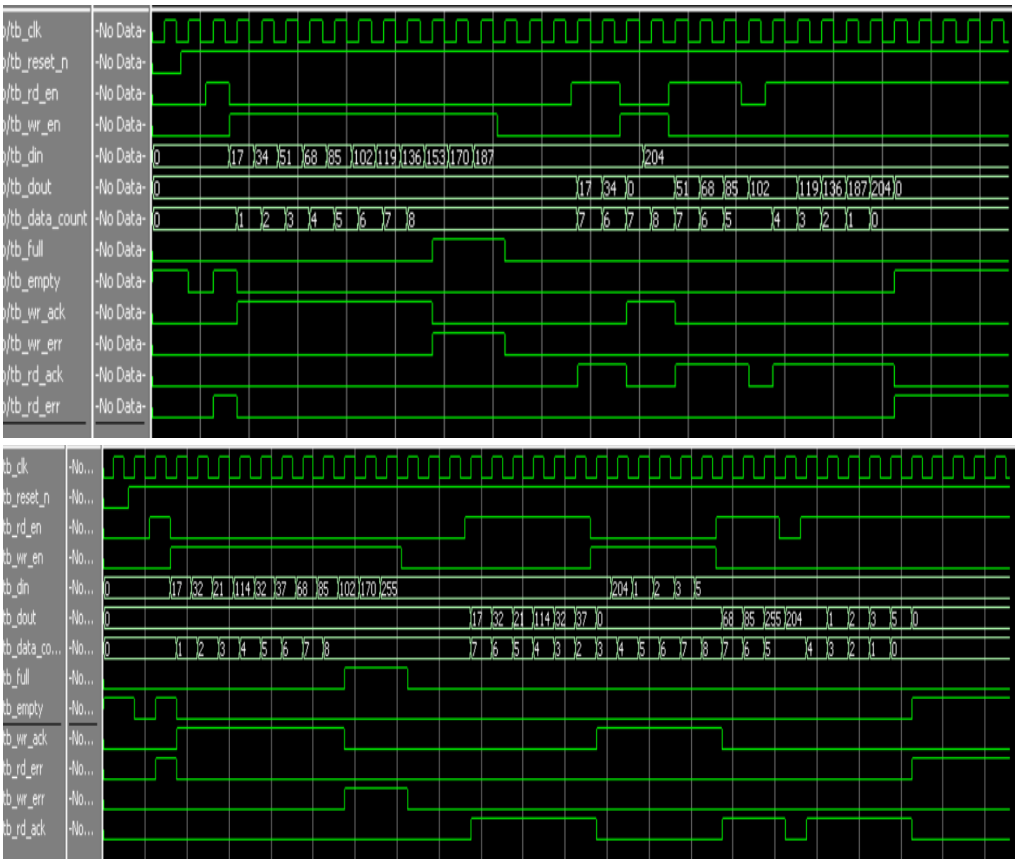
| Module 이름 | 구분 | 이름 | 비트 수 | 설명 |
|---|---|---|---|---|
| fifo | input | clk | 1-bit | Clock |
| | | reset_n | 1-bit | Active low에 동작하는 reset 신호 |
| | | rd_en | 1-bit | Write enable |
| | | wr_en | 1-bit | Write address |
| | | d_in | 32-bits | Read address |
| | output | d_out | 32-bits | Read data |
| | | full | 1-bit | Data full signal |
| | | empty | 1-bit | Data empty signal |
| | | wr_ack | 1-bit | Write acknowledge |
| | | wr_err | 1-bit | Write error |
| | | rd_ack | 1-bit | Read acknowledge |
| | | rd_err | 1-bit | Read error |
| | | data_count | 4-bits | Data count vector |
| fifo_ns | input | wr_en | 1-bit | Write enable |
| | | rd_en | 1-bit | Read enable |
| | | state | 3-bits | Current state |
| | | data_count | 4-bits | Data count vector |
| | output | next_state | 3-bits | Next state |
| fifo_cal_addr | input | state | 3-bits | Current state |
| | | head | 3-bits | Current head pointer |
| | | tail | 3-bits | Current tail pointer |
| | | data_count | 4-bits | Current data count vector |
| | output | we | 1-bit | Register file write enable |
| | | re | 1-bit | Register file read enable |
| | | next_head | 3-bits | Next head pointer |
| | | next_tail | 3-bits | Next tail pointer |
| | | next_data_count | 4-bits | Next data count vector |
| fifo_out | input | State | 3-bits | Current state |
| | | Data_count | 4-bits | Current data count vector |
| | output | Full | 1-bit | Data full signal |
| | | Empty | 1-bit | Data empty signal |
| | | Wr_ack | 1-bit | Write acknowledge |
| | | Wr_err | 1-bit | Write error |
| | | Rd_ack | 1-bit | Read acknowledge |
| | | Rd_err | 1-bit | Read error |

해당 FIFO는 전에 구현한 Register File을 기반으로 만들게 되었으며 FSM을 통해 next state를 결정하는 모듈과 계산을 위한 Calculator 모듈, output을 결정하는 output 모듈을 기반으로 만들게 된다. 나머지 모듈들은 해당 동작을 해주기위한 sub module의 모음이다. 세부적인 FSM의 동작은 원리에서 FSM을 소개할 때 같이 올려 두었습니다.

## 4. 설계 검증 및 실험 결과

### A. 시뮬레이션 결과

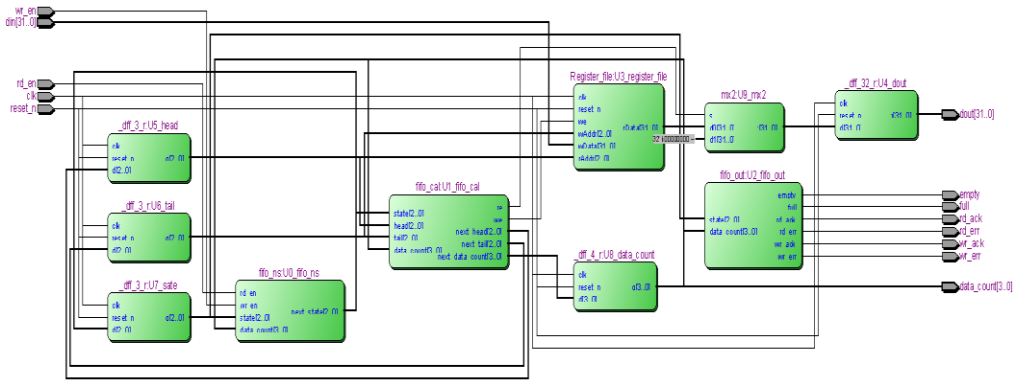
FIFO




해당 결과를 살펴보면 FIFO의 규칙을 가지고 있는 것을 확인 할 수 있다. Reset으로 값이 시작되며 wr_en의 값이 rising되면 din의 값을 저장하는 것을 확인 할 수 있으며 최대 저장가능한 정보의 개수는 8개이므로 count의 값이 8이 되면 Full의 값이 1로 rising되며 다음 데이터들은 저장이 되지 않는 것을 확인 할 수 있다. 다음으로는 저장된 정보를 읽어오는 과정으로 값을 읽을 때는 가장 먼저 저장되는 값이 먼저 불려오는 FIFO의 구조를 나타내고 있다. 정보를 읽는 동작을 수행하기 위해서는 rd_en이 값을 rising시키면 된다. 만약 읽어오는 정보의 개수가 0이되면 empty의 값이 rising하는 것을 확인 할 수 있다.

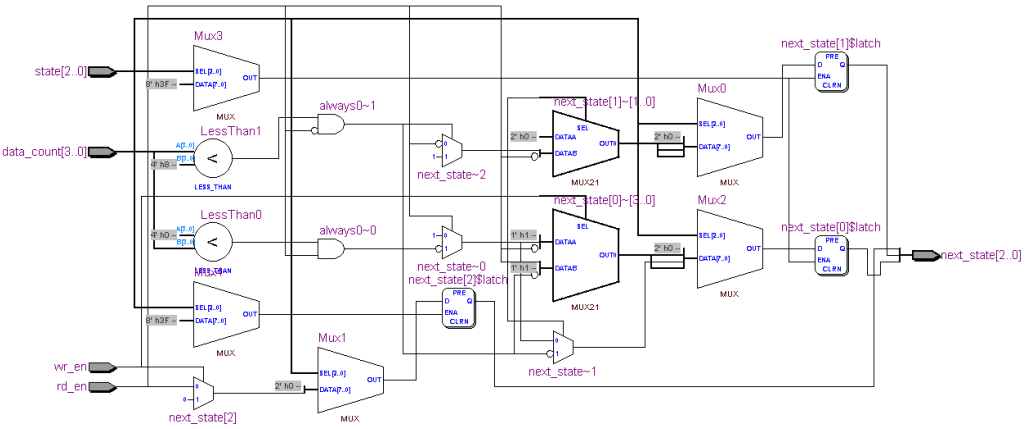
B. 합성(synthesis) 결과

RTL Viewer

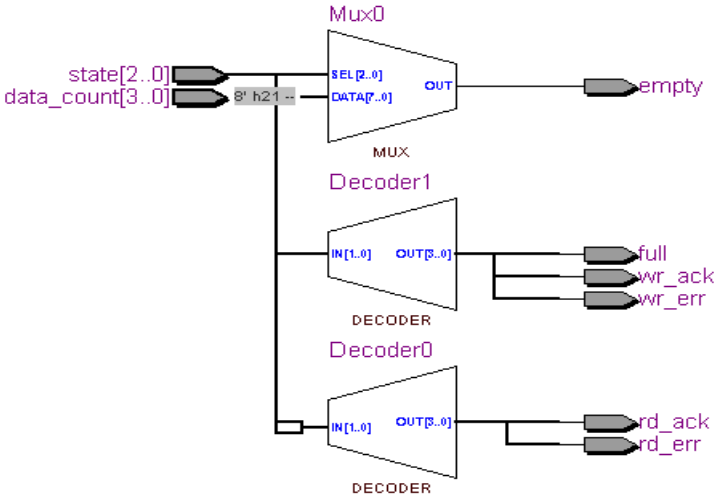


FIFO의 전체적인 모습이다. 해당 사진에는 ns, out, cal에 관련된 모듈을 간단하게 확인할 수 있다. 해당 모듈의 갈무리한 모습은 아래와 같다.

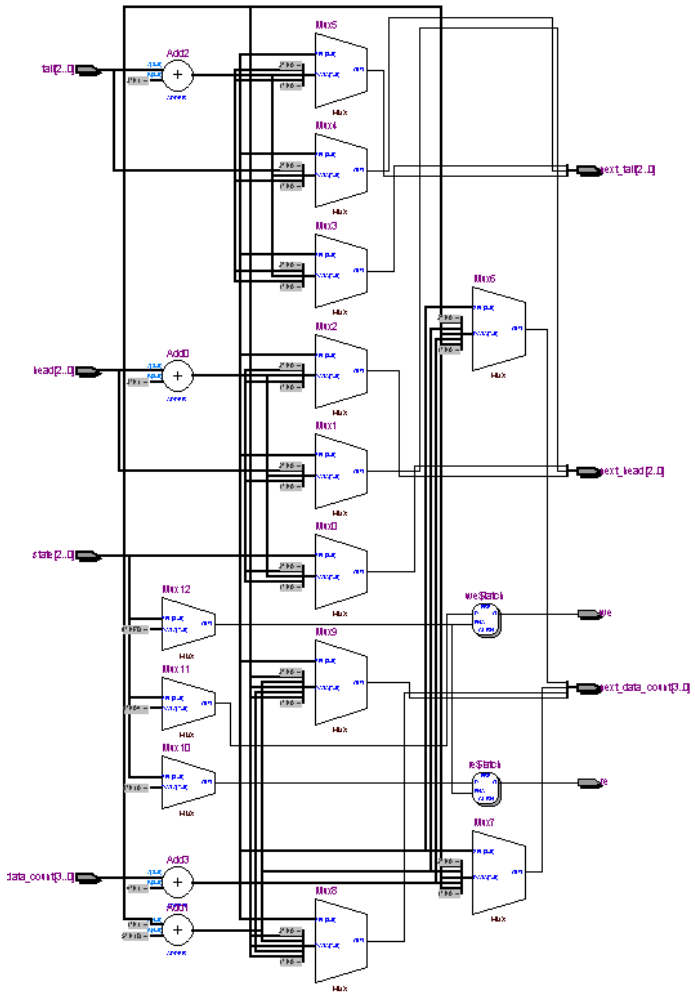
Next state




Out



Calculator



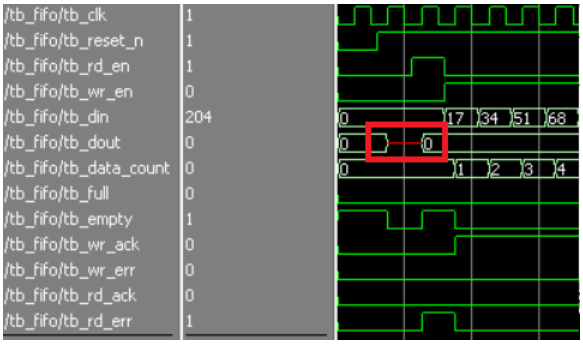
## Flow summary

| Flow Summary | |
|---|---|
| Flow Status | Successful - Tue Oct 23 13:06:01 2018 |
| Quartus II 64-Bit Version | 13.0.0 Build 156 04/24/2013 SJ Web Edition |
| Revision Name | fifo |
| Top-level Entity Name | fifo |
| Family | Cyclone II |
| Device | EP2C70F896C6 |
| Timing Models | Final |
| Total logic elements | 331 / 68,416 ( < 1 % ) |
| Total combinational functions | 203 / 68,416 ( < 1 % ) |
| Dedicated logic registers | 301 / 68,416 ( < 1 % ) |
| Total registers | 301 |
| Total pins | 78 / 622 ( 13 % ) |
| Total virtual pins | 0 |
| Total memory bits | 0 / 1,152,000 ( 0 % ) |
| Embedded Multiplier 9-bit elements | 0 / 300 ( 0 % ) |
| Total PLLs | 0 / 4 ( 0 % ) |

## 5. 고찰 및 결론

### A. 고찰

해당 실습을 진행하면서 mux의 예외처리에 따른 다른 결과 값을 확인했습니다. 결론적으로는 Don't care의 값이 들어오게 되면 0의 값을 선택하는 것이 맞는 방식이지만 내용을 확인하지 못하고 Don't care의 값이 들어오면 mux의 결과가 Don't care로 출력 될 수 있도록 설계하였습니다. 그 결과 dout의 값이 en과 관련된 신호가 없으면 Don't care의 값을 유지하고 있는 것을 확인 할 수 있었습니다.



<<(mux가 0의 값을 출력하지 않아서 발생한 문제)

해당 문제는 원래 설계했던 mux의 예외처리를 바꾸어 주면서 해결하게 되었습니다. 즉 선택 값이 1을 제외한 나머지 경우 don't care와 같은 다른 값을 포함하는 것이 아니라 0의 값을 가질 수 있도록 재설계 하였습니다.

### B. 결론

실습을 통하여 FIFO에 대한 구조를 좀더 명확히 하는 기회가 되었습니다. 전에 실습했던 Register File에 관련된 동작도 다시 점검하며 구현한 module에 대한 이해와 개념을 다시 확인하는 실습이었습니다. 특히 실습에 사용된 FIFO의 구조를 다른 영역에서도 많이 사용하며 응용하고 있습니다. 또한 FIFO를 배우면서 LIFO도 같이 알아보게 되었습니다. LIFO방식은 Last in first out으로 가장 마지막에 들어간 정보가 가장 먼저 나오는 방식입니다. FIFO랑은 반대의 개념이라고 생각하면 쉽습니다. 이 구조를 Stack이라고 하는데 해당 자료구조를 응용하면 홈페이지에서 뒤로 가기 앞으로 가기 등의 기능을 예로 제시할 수 있습니다. 또한 이번에 배운 FIFO는 컴퓨터와 관련된 영역만이 아니라 다른 영역에서도 많이 사용되는데 예로 들면 식품을 유통하는 데 있어 가장 먼저 들어온 식품은 사용될 때 먼저 사용하며 식품의 신선함을 유지하는데 사용하는 방식도 FIFO에서 비롯된 방식입니다. 컴퓨터 상에서는 데이터가 먼저 들어오면 가장 먼저 해당 데이터를 처리해주는 식으로 나머지 데이터들은 대기하고 있는 형태를 말할 때 사용하기도 합니다. 특히 queue라도 불리는 FIFO의 구조는 자료구조의 이진 트리를 배우면서 레벨 순회자를 배울 때도 응용되며 많은 곳에서 사용되고 있음을 알 수 있었습니다.

## 6. 참고문헌

김영민/FIFO/광운대학교/2018

김영민/FIFO_new/광운대학교/2018

김영민/Finite State Machine/2018

김영민/Register Files/2018

FIFO의 개념/https://en.wikipedia.org/wiki/FIFO

LIFO/https://www.investopedia.com/terms/l/lifo.asp

Queue/https://terms.naver.com/entry.nhn?docId=834442&cid=42344&categoryId=42344

Stack/https://terms.naver.com/entry.nhn?docId=2837556&cid=40942&categoryId=32841