

# 컴퓨터 공학 기초 실험2 보고서

실험제목: Ripple-Carry Adder (RCA)

실험일자: 2022년 09월 20일 (화)

제출일자: 2022년 09월 20일 (화)

학 과: 컴퓨터정보공학부

담당교수: 공영호 교수님

실습분반: 화요일 0,1,2

학 번: 2019202050

성 명: 이강현

## 1. 제목 및 목적

### A. 제목

Ripple-Carry Adder (RCA)

### B. 목적

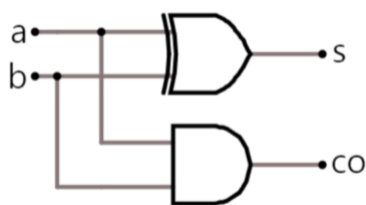
1비트 입력을 두개 받아 합과 캐리를 출력해주는 half adder와 2개의 1비트 입력과 1개의 1비트 캐리를 입력받아 합과 캐리를 출력하는 full adder, 그리고 full adder를 사용하여 ripple carry adder를 구현한다.

## 2. 원리(배경지식)

컴퓨터에서 수를 표현하는 방법 중 sign/magnitude 방법과 two's complement 방법이 있는데 sign/magnitude는 0을 4비트를 기준으로 하였을 때 맨 앞자리 즉 MSB는 부호만을 나타내므로 0000이나 1000이나 모두 0을 나타내고 덧셈시 오버플로우같은 문제점이 발생할 수 있어 컴퓨터에서는 two's complement를 사용한다. 덧셈에서 문제점을 해결하기 위해 2의 보수라는 개념을 사용하는데 주어진 수보다 한자리가 더 높고 가장 높은 자릿수가 1인 수에서 2의 제곱수에서 주어진 수를 빼면 그 수의 보수를 구할 수 있다. 쉬운 방법으로는 주어진 수의 모든 자리를 invert하고 1을 더해주는 방식으로 2의 보수를 구할 수 있다.

그럼 다음으로 이번 실험에서 구현할 가산기들에 대해 알아본다.

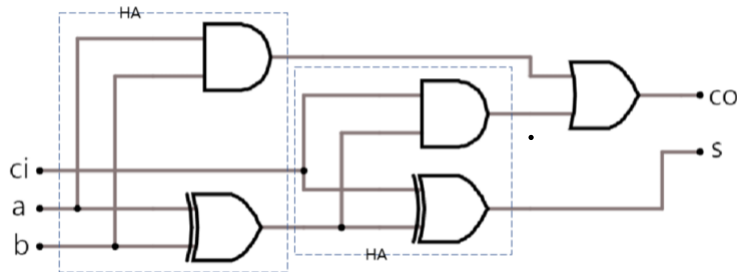
### <half adder>



논리회로는 왼쪽과 같으며

1비트 입력값 a,b를 가지고 합의 값인 s와 캐리 발생여부인 CO를 출력값으로 가지는 가산기이다.

### <full adder>

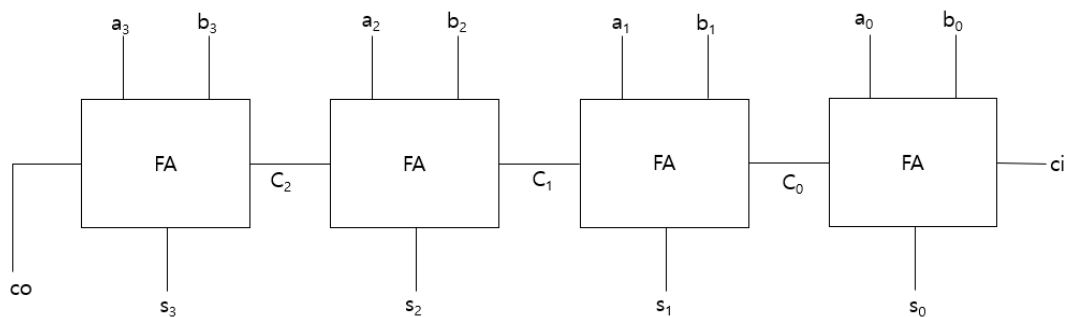


논리 회로는 왼쪽과

같다. 논리식 그대로 구현할 수도 있고 half adder 두개와 or게이트 하나를 사용하여 구현할 수 있다.

Half adder와는 다르게 carry in이라는 입력값을 추가로 가진다는 특성이 있다.

### <4-bit ripple carry adder>



4비트 가산기이다 full adder 4개를 이어붙여 구현하며 낮은 자리수 비트부터 캐리값을 넘겨주며 계산한다. 여러 개를 이어붙여 더 높은 비트수를 가진 가산기를 만들 수 있으나 앞의 회로는 뒤에서부터 값을 받아오므로 처리속도가 느리다는 특징이 있다.

### 3. 설계 세부사항

## 1. Half adder

| Input |   | Output |   |
|-------|---|--------|---|
| A     | B | Co     | s |
| 0     | 0 | 0      | 0 |
| 0     | 1 | 0      | 1 |
| 1     | 0 | 0      | 1 |
| 1     | 1 | 1      | 0 |

| a \ b | 0 | 1        |
|-------|---|----------|
| 0     | 0 | 0        |
| 1     | 0 | <b>1</b> |

$$co = ab$$

<카르노맵을 통한 co 논리식 도출>

| a \ b | 0        | 1        |
|-------|----------|----------|
| 0     | 0        | <b>1</b> |
| 1     | <b>1</b> | 0        |

$$s = a \oplus b$$

<카르노맵을 통한 s 논리식 도출>

Half adder에서 캐리는 입력값이 모두 1일 때 1이고 합은 둘 중 하나만 1일때 1이다. 따라서 위와 같은 논리식과 진리표를 가진다.

## 2. Full adder

| Input |   |    | Output |    |
|-------|---|----|--------|----|
| A     | B | Ci | S      | Co |
| 0     | 0 | 0  | 0      | 0  |
| 0     | 0 | 1  | 1      | 0  |
| 0     | 1 | 0  | 1      | 0  |
| 0     | 1 | 1  | 0      | 1  |

|   |   |   |   |   |
|---|---|---|---|---|
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

진리표는 위와 같다.

Half adder와 달리 carry in이 있기 때문에 논리식은 아래와 같다.

| Ci \ ab | 00 | 01 | 11 | 10 |
|---------|----|----|----|----|
| 0       | 0  | 1  | 0  | 1  |
| 1       | 1  | 0  | 1  | 0  |

$$s = \bar{c}i\bar{a}b + \bar{c}ia\bar{b} + ci\bar{a}\bar{b} + ciab$$

<카르노맵을 통한 s 논리식 도출>

| Ci \ ab | 00 | 01 | 11 | 10 |
|---------|----|----|----|----|
| 0       | 0  | 0  | 1  | 0  |
| 1       | 0  | 1  | 1  | 1  |

$$co = ab + cia + cib$$

<카르노맵을 통한 co 논리식 도출>

### 3. 4-bit ripple carry adder

이번 실험에서는 4비트 ripple carry adder를 구현할 것이므로 full adder 4개가 연결되어 있으며 입력값 a,b 출력값 s는 4비트이고 중간에 캐리를 건네주는 wire는 3비트로 구성한다. 입력값 ci와 출력값 co는 1비트로 구성되어 있다.

<Gates.v 구현>

설계에서 사용할 논리게이트 inverter, 2input nand, 2input and, 2input or, 2input xor등이 구현되어 있다.

#### <ha.v 구현>

Half adder가 구현되어 있다. gate.v에 구현된 함수를 이용하여 구현했다.

#### <fa.v 구현>

Full adder가 구현되어 있다. gate.v에 구현된 함수와 ha.v에 구현된 half adder 2개를 이용하여 구현하였다.

#### <rca4.v 구현>

fa.v에 구현된 full adder 4개를 이어붙여 rca4(4-bit ripple carry adder)를 구현했다.

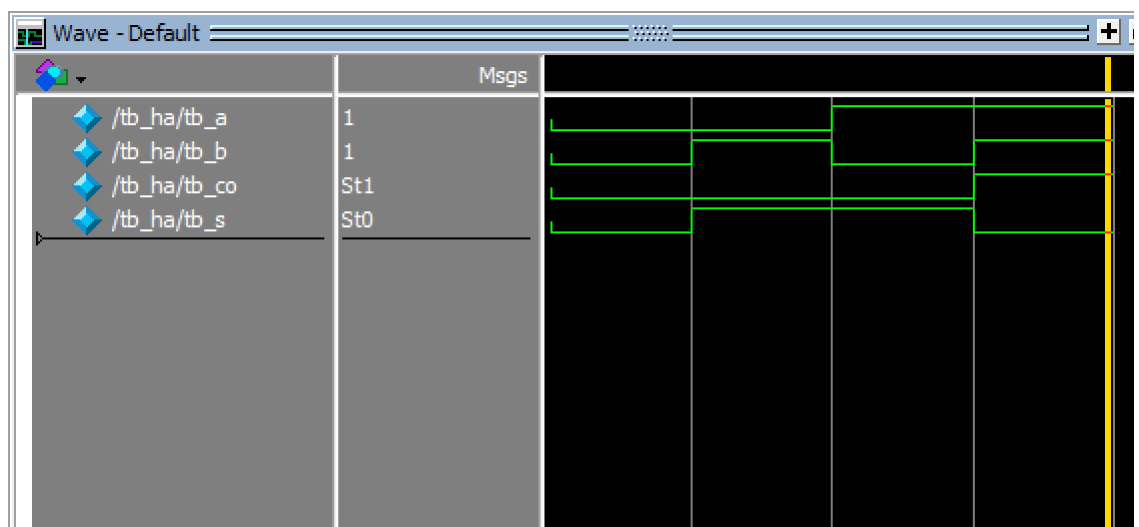
#### <각종 .tb파일>

ha.v, fa.v, rca4.v파일들의 테스트벤치들로 값의 전달유무와 회로가 정상적으로 작동하는지 확인하기 위해 만들었으며 특이사항으로는 rca4구현시 입력 값이 9개이므로 exhaustive verification을 사용하면 경우의 수가 너무 많으므로 tb\_rca4.v 파일은 directed verification을 실시하였다.

### 4. 설계 검증 및 실험 결과

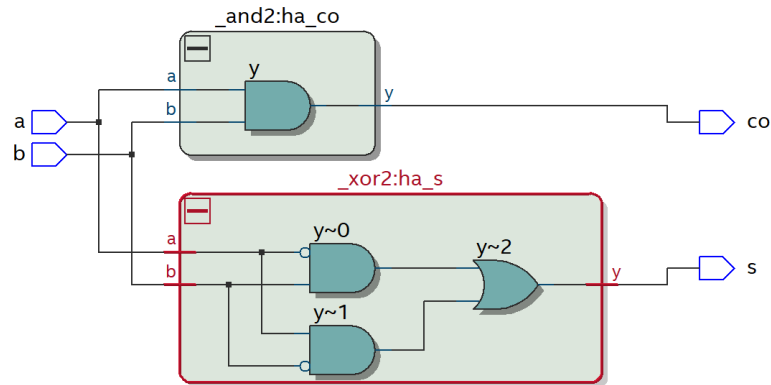
#### <half adder>

##### A. 시뮬레이션 결과



Half adder의 시뮬레이션 결과이다. 00~11까지의 입력값을 주었다.  
결과가 잘 나온 것을 확인할 수 있다.

## B. 합성(synthesis) 결과



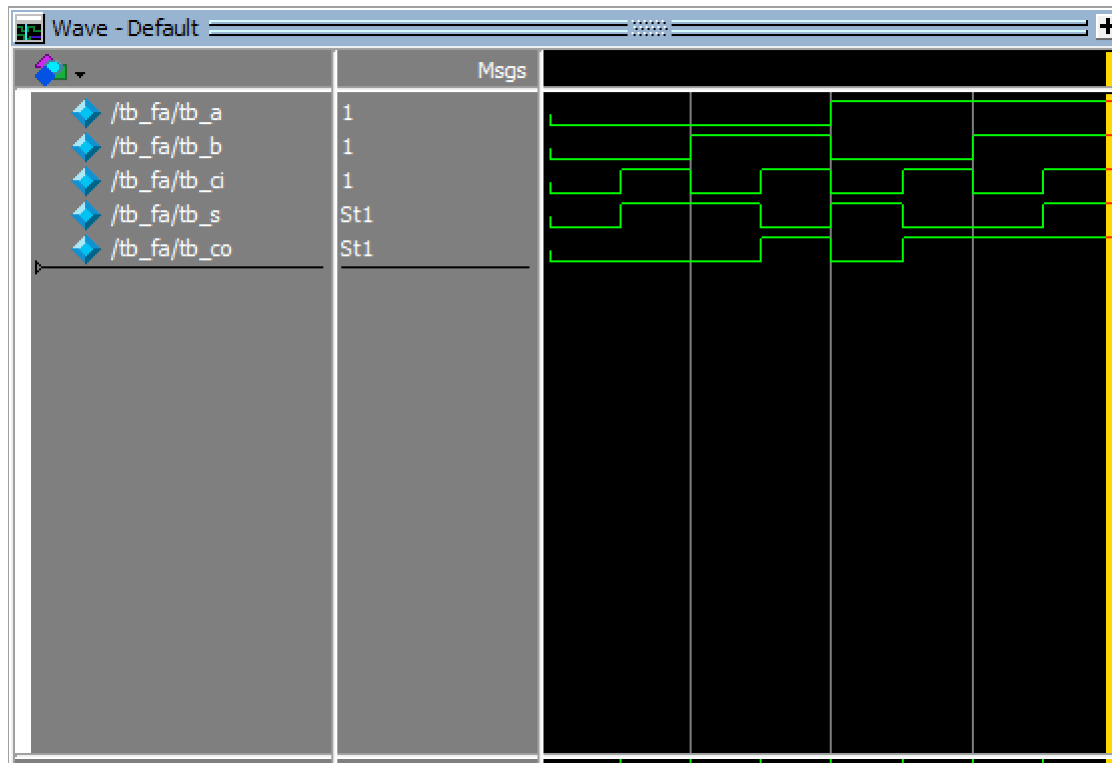
Xor2은 gate.v에 있는 inverter and or게이트를 사용하여 직접 구현하였다. 논리식에 맞는 회로가 구현되었다.

| Flow Summary                    |   |
|---------------------------------|---|
| <<Filter>>                      |   |
| Flow Status                     | Successful - Tue Sep 20 13:48:31 2022       |
| Quartus Prime Version           | 18.1.0 Build 625 09/12/2018 SJ Lite Edition |
| Revision Name                   | ha  |
| Top-level Entity Name           | ha  |
| Family                          | Cyclone V                                   |
| Device                          | 5CSXFC6D6F31C6                              |
| Timing Models                   | Final                                       |
| Logic utilization (in ALMs)     | N/A   |
| Total registers                 | 0   |
| Total pins                      | 4   |
| Total virtual pins              | 0   |
| Total block memory bits         | 0   |
| Total DSP Blocks                | 0   |
| Total HSSI RX PCSs              | 0   |
| Total HSSI PMA RX Deserializers | 0   |
| Total HSSI TX PCSs              | 0   |
| Total HSSI PMA TX Serializers   | 0   |
| Total PLLs                      | 0   |
| Total DLLs                      | 0   |

Flow summary 사진이다. Successful로 잘 컴파일 되었다.

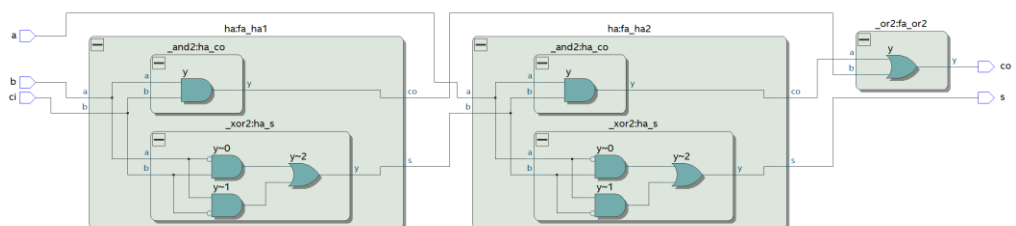
<full adder>

## A. 시뮬레이션 결과



Full adder의 시뮬레이션 결과이다. 000~111까지 논리식에 맞게 잘 waveform이 형성되었다.

## B. 합성(synthesis) 결과



회로가 half adder 2개와 or 게이트 1개로 구현되어있는 것을 눈으로 확인할 수 있다.



## Flow Summary

<<Filter>>











|                                 |   |
|---------------------------------|---|
| Flow Status                     | Successful - Tue Sep 20 13:53:57 2022       |
| Quartus Prime Version           | 18.1.0 Build 625 09/12/2018 SJ Lite Edition |
| Revision Name                   | fa  |
| Top-level Entity Name           | fa  |
| Family                          | Cyclone V                                   |
| Device                          | 5CSXFC6D6F31C6                              |
| Timing Models                   | Final                                       |
| Logic utilization (in ALMs)     | N/A   |
| Total registers                 | 0   |
| Total pins                      | 5   |
| Total virtual pins              | 0   |
| Total block memory bits         | 0   |
| Total DSP Blocks                | 0   |
| Total HSSI RX PCSs              | 0   |
| Total HSSI PMA RX Deserializers | 0   |
| Total HSSI TX PCSs              | 0   |
| Total HSSI PMA TX Serializers   | 0   |
| Total PLLs                      | 0   |
| Total DLLs                      | 0   |

Flow summary 또한 잘 실행되었음을 알 수 있다.

## <4-bits ripple carry adder>

### A. 시뮬레이션 결과

#### Binary results

| Wave - Default  |  |       |       |       |      |       |       |       |       |       |  |
|---|--|-------|-------|-------|------|-------|-------|-------|-------|-------|--|
|   |  | Msgs  |       |       |      |       |       |       |       |       |  |
|  |  /tb_rca4/tb_a      | 0111  | 0001  | 0010  | 0001 | 0111  | 0000  | 1111  | 1001  | 0111  |  |
|  |  /tb_rca4/tb_b      | 1111  | 0001  | 0010  | 0001 | 0000  | 0111  | 1000  | 1111  |       |  |
|   |  /tb_rca4/tb_ci     | 1     |       |       |      |       |       |       |       |       |  |
|   |  /tb_rca4/tb_co     | St1   |       |       |      |       |       |       |       |       |  |
|  |  /tb_rca4/tb_s      | 0111  | 0010  | 0011  | 0011 | 1001  | 0001  | 0111  | 0001  | 0111  |  |
|  |  /tb_rca4/tb_result | 10111 | 00010 | 00011 |      | 01001 | 00001 | 10111 | 10001 | 10111 |  |

## Decimal results

| Variable           | Value |
|--------------------|-------|
| /tb_rca4/tb_a      | 7     |
| /tb_rca4/tb_b      | -1    |
| /tb_rca4/tb_ci     | -1    |
| /tb_rca4/tb_co     | 1     |
| /tb_rca4/tb_s      | 7     |
| /tb_rca4/tb_result | -9    |

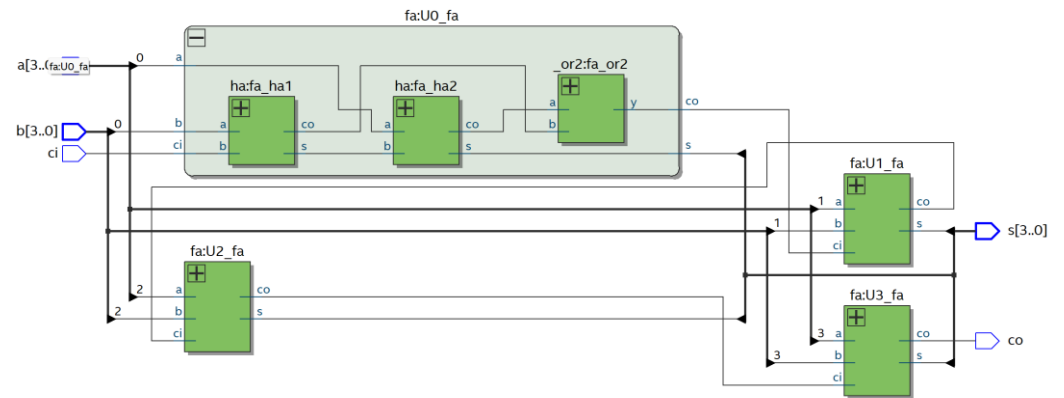
## Unsigned results

| Variable           | Value |
|--------------------|-------|
| /tb_rca4/tb_a      | 7     |
| /tb_rca4/tb_b      | 15    |
| /tb_rca4/tb_ci     | 1     |
| /tb_rca4/tb_co     | 1     |
| /tb_rca4/tb_s      | 7     |
| /tb_rca4/tb_result | 23    |

입력값이 총 9개이므로 Directed verification을 사용하여 testbench를 진행하였다. 캐리가 발생하지 않는 경우(ex 0001과 0010과 0연산), 캐리가 발생하는 경우(ex 0001과 0001과 0연산), 캐리와 합이 모두 1인 연산이 나타나는 경우(ex 0111과 0001과 1연산) 입력값 a,b가 모두 0인 경우(ex 0000과 0000과 1 연산) 4비트 범위를 벗어나는 경우(ex 1111과 0111과 1연산)을 다양한 경우를 고려해서 테스트 값을 설정하였다.

Decimal의 수 표현범위는 -8~7이다(two's complement이므로)따라서 7과 1을 더하고 ci는 1인경우에 값은 9이므로 decimal에선 범위를 넘어섰기 때문에 1001(-7로 나타나지만) radix를 unsigned로 바꿨을 때는 0~15까지 표현 가능하므로 결과값이 9로 올바르게 나타난다. 값을 올바르게 처리한다는 것을 알 수 있다.

## B. 합성(synthesis) 결과



Fa 모듈 4개를 사용하는 것을 볼 수 있다.

| Flow Summary                    |   |
|---------------------------------|---|
| <<Filter>>                      |   |
| Flow Status                     | Successful - Tue Sep 20 13:58:42 2022       |
| Quartus Prime Version           | 18.1.0 Build 625 09/12/2018 SJ Lite Edition |
| Revision Name                   | RCA   |
| Top-level Entity Name           | rca4  |
| Family                          | Cyclone V                                   |
| Device                          | 5CSXFC6D6F31C6                              |
| Timing Models                   | Final                                       |
| Logic utilization (in ALMs)     | N/A   |
| Total registers                 | 0   |
| Total pins                      | 14  |
| Total virtual pins              | 0   |
| Total block memory bits         | 0   |
| Total DSP Blocks                | 0   |
| Total HSSI RX PCSs              | 0   |
| Total HSSI PMA RX Deserializers | 0   |
| Total HSSI TX PCSs              | 0   |
| Total HSSI PMA TX Serializers   | 0   |
| Total PLLs                      | 0   |
| Total DLLs                      | 0   |

Flow summary 또한 정상적으로 결과를 내놓았다.

## 5. 고찰 및 결론

### C. 고찰

쿼터스에서 4'b0001이나 4'hf와 같이 값을 나타내는 방법이나 입력 값이 너무 많을 때 어떻게 회로를 테스트 하는지에 대한 방법을 알았다. 또한 그러한 테스트벤치를 설계하는 법에 대해 배울 수 있는 실험이었다. 또한 설계하던 중 시뮬레이션 툴이 잘 작동하지 않는 문제점이 생겼는데 이는 모듈의 이름을 올바르게 작성하지 않아 모듈이 로드되지 않아 생긴 문제였다. 설계를 하는 과정에서 이러한 사소한 실수들이 결함을 만든다는 것을 체감할 수 있는 오류였다. 그리고 1비트가 아닌 n비트를 활용하여 입력값을 설정하는 첫 실험이라 이론적인 것을 학습하는데 도움이 되었고 다음에 쉽게 활용할 수 있을 것 같다.

### D. 결론

RCA를 설계하는 과정에서 여러가지 개념들이 사용되었다. Half adder에서 시작하여 full adder 그리고 4-bit ripple carry adder까지 결국엔 작은 모듈로 큰 모듈을 만드는 과정이었다. 오류없이 이러한 작업을 해나가기 위해서는 작은 모듈들을 결함없이 안전하게 만드는 것이 꼭 필요한 작업이며 그러기 위해서는 논리식이나 설계모델을 확실히 정리해둔 후 설계해야 함을 배우는 시간이었다.

32-bit RCA를 설계하는 방법은 단순히 입력값을 32비트화 하고 full adder를 이어붙이는 방법이 있을 것이다. 하지만 RCA는 순차적으로 이루어지는 연산이므로 처리속도가 늦다는 단점이 있는데 이를 해결하는 방법은 full adder를 나눠서 병렬로 처리하는 방법이 있을 것 같다.

## 6. 참고문헌

공영호 교수님/컴퓨터공학기초실험2/2022