13주차 예비보고서

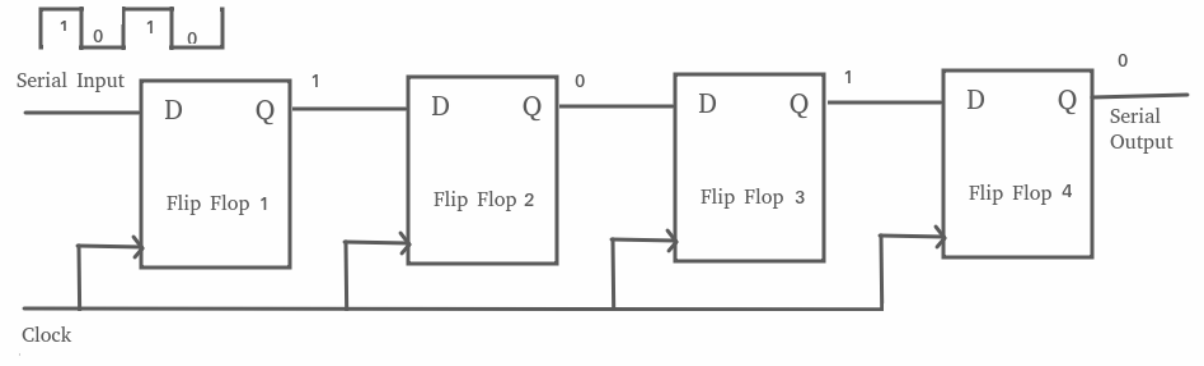
전공: 컴퓨터공학 학년: 2학년 학번: 20191629 이름: 이주헌

1. Shift register에 대해 조사하시오.

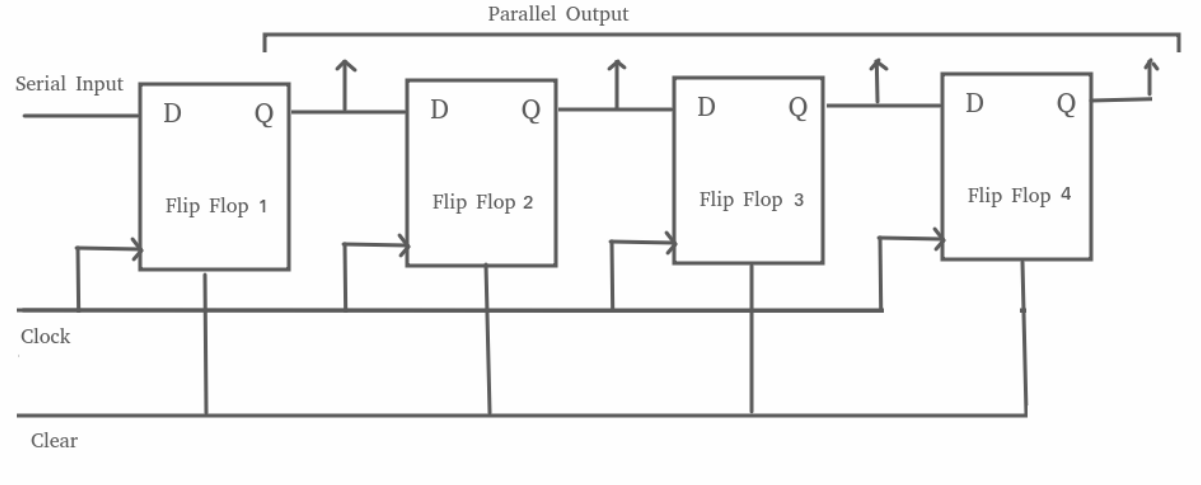
시프트 레지스터는 플립플롭을 연결하여 만든 회로로, 클럭의 한 주기가 지날 때마다 1비트 데이터가 한 방향으로 “이동(shift)”하는 회로를 말한다. 시프트 레지스터의 입력과 출력은 각각 직렬과 병렬으로 나눌 수 있다. 따라서, 시프트 레지스터는 다음과 같은 총 네 가지의 종류로 나눌 수 있다.

- 직렬-직렬 시프트 레지스터 (serial-serial shift register, SISO)
- 직렬-병렬 시프트 레지스터 (serial-parallel shift register, SIPO)
- 병렬-직렬 시프트 레지스터 (parallel-serial shift register, PISO)
- 병렬-병렬 시프트 레지스터 (parallel-parallel shift register, PIPO)

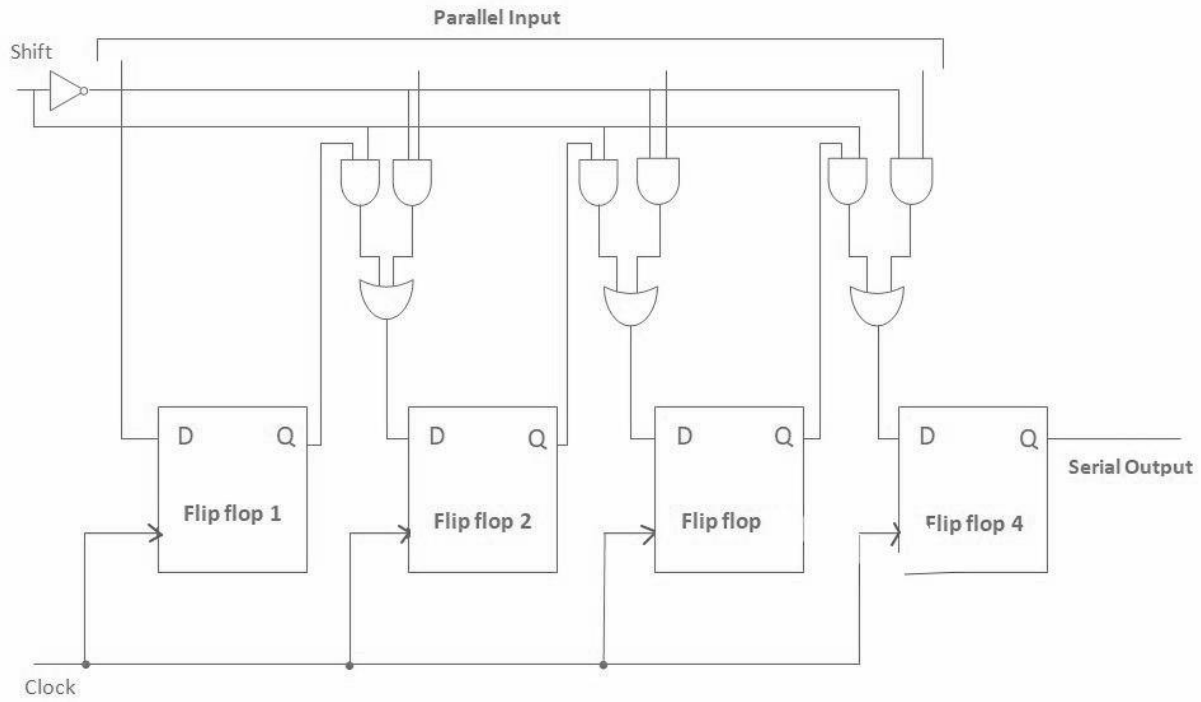
직렬은 한 번에 하나의 입력을 받거나 하나만을 출력을 한다는 의미이고, 병렬은 한 번에 여러 개의 입/출력을 한다는 의미이다. 여기서 직렬-직렬 시프트 레지스터는 다음과 같은 회로를 통해 구현할 수 있다.



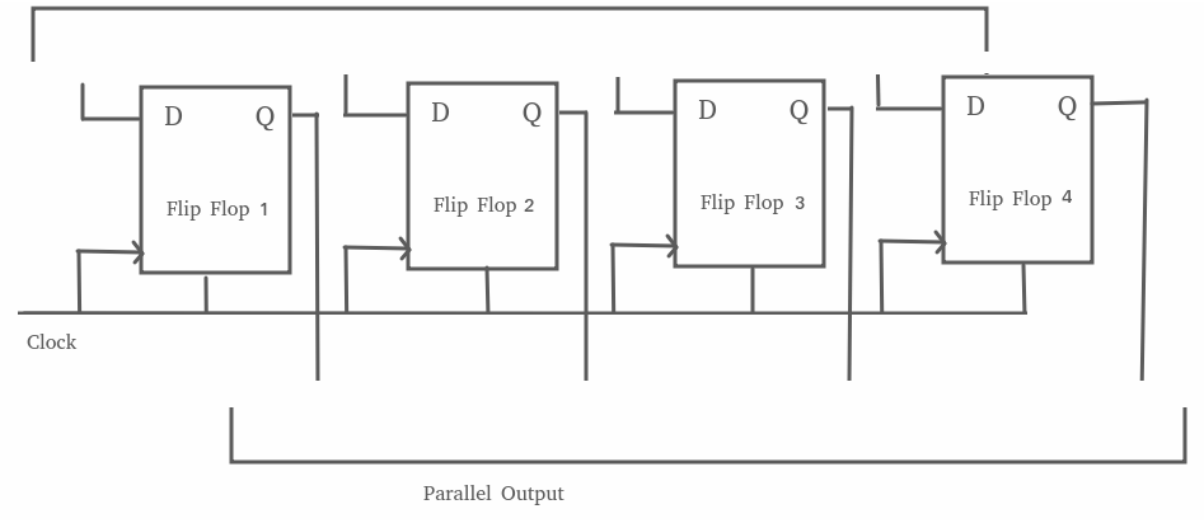
직렬 입력을 받아 병렬적으로 출력하는 회로 역시 직렬-직렬 시프트 레지스터를 바탕으로 다음과 같이 구성할 수 있다.



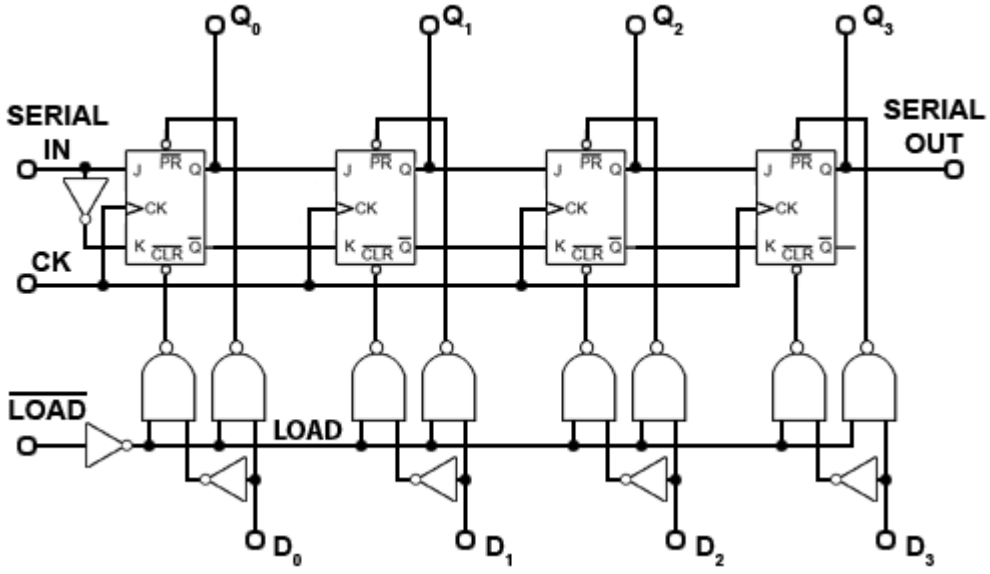
병렬-직렬 시프트 레지스터는 병렬 데이터를 직렬 데이터로 변환할 때 사용되는데, 다음과 같은 회로를 바탕으로 구성된다.



마지막으로 병렬-병렬 시프트 레지스터는 병렬 신호를 직렬 신호로 변환할 필요가 없기 때문에 다음과 같이 플립플롭 사이의 의존 관계가 존재하지 않는다.

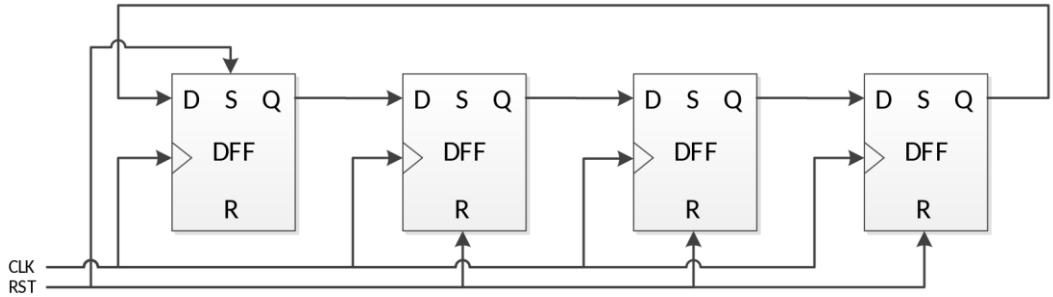


이 모든 종류의 시프트 레지스터를 한 번에 구현하기 위해서는 다음과 같은 회로를 사용하여 구현할 수 있다.

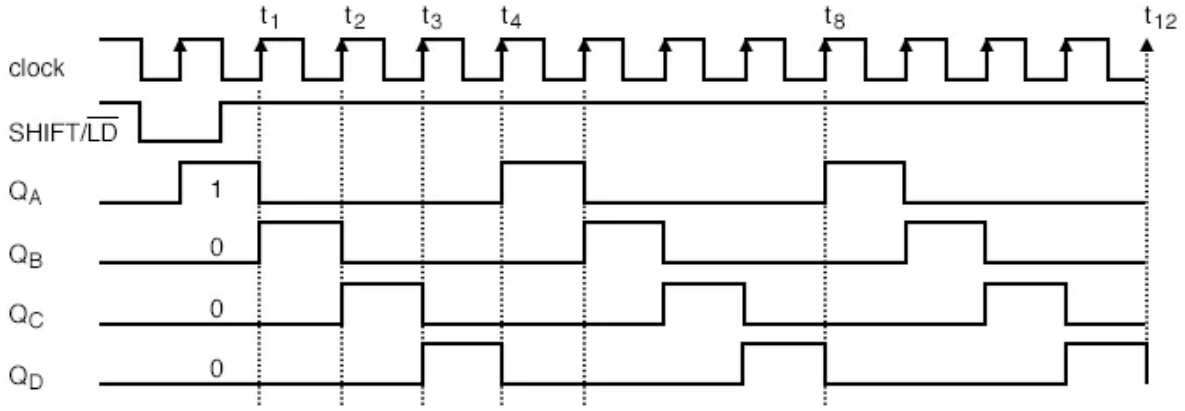


2. Ring counter에 대해 조사하시오.

링 계수기는 시프트 레지스터를 이용해서 구현할 수 있는 간단한 형태의 계수기이다. 링 계수기는 한 번에 하나의 출력을 내기 때문에 one-hot 계수기로 불리기도 한다. 링 계수기의 회로도는 다음과 같다.



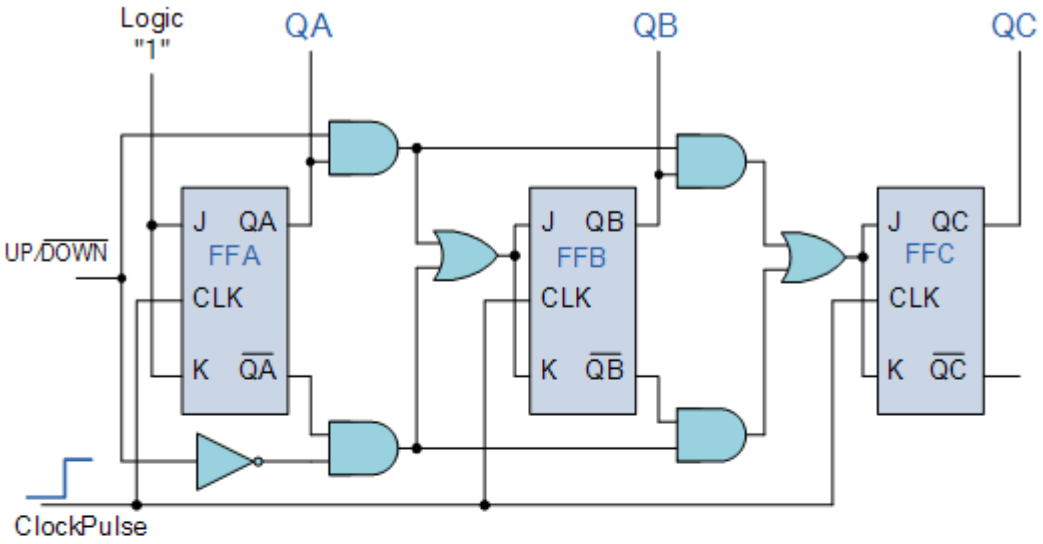
위 회로를 초기화하면 1000으로 초기화되게 되는데, 이 상태에서 클럭 신호를 주게 되면 한 칸씩 데이터가 오른쪽으로 이동하게 된다. 만약 데이터가 오른쪽 끝에 도달하게 되면, 다시 1000으로 초기화되어 위와 같은 패턴을 반복하게 된다. 이를 그래프로 나타내면 다음과 같은 그래프를 볼 수 있다.

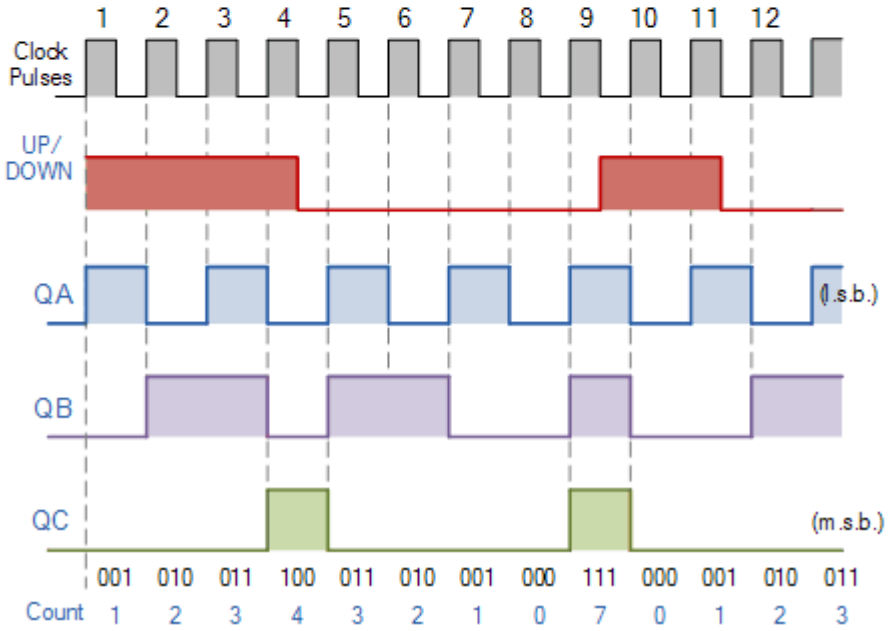


Load 1000 into 4-stage ring counter and shift

3. Up-down counter에 대해 조사하시오.

상하향 계수기는 지난 실험에서 구현한 계수기에 숫자를 반대로 세는 기능을 추가한 계수기이다. 일반 계수기와 달리 상하향 계수기는 숫자를 정방향으로 셀 지, 역방향으로 셀 지 결정하는 입력 핀이 존재한다. 만약 상향으로 숫자를 센다면 현재 숫자에서 1을 더하는 방식으로 수를 세고, 그렇지 않다면 현재 숫자에서 1을 빼는 방식으로 수를 세게 된다. 다음은 T 플립플롭을 사용하여 상하향 계수기를 구현한 것이다.



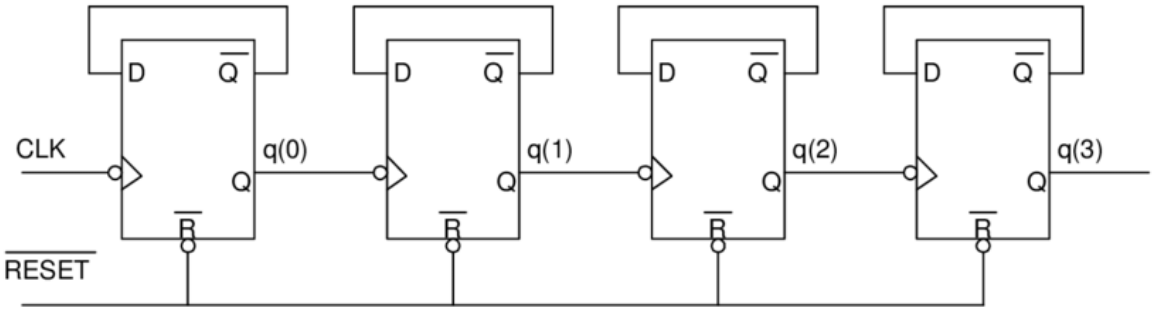


위 회로는 상/하향 신호를 T 플립플롭으로 들어가는 입력 신호와 합성하여 플립플롭의 동작 방식을 변경하는 것으로 동작한다.

4. Ripple counter에 대해 조사하시오.

Ripple 계수기는 ripple carry adder에서의 그것과 마찬가지로, 이전 플립플롭의 결과값 신호를 이용하여 또 다른 플립플롭을 활성화시키는 계수기이다. 즉, 하나의 클럭 신호를 공유하지 않고, 클럭 신호는 단 하나의 플립플롭에 입력되며, 해당 플립플롭의 출력값이 다른 플립플롭을 조작한다는 것이다. 따라서, ripple 계수기는 비동기 계수기의 한

종류라고 생각해도 무방하다. 다음은 D 플립플롭을 사용하는 간단한 ripple 계수기의 예시이다.

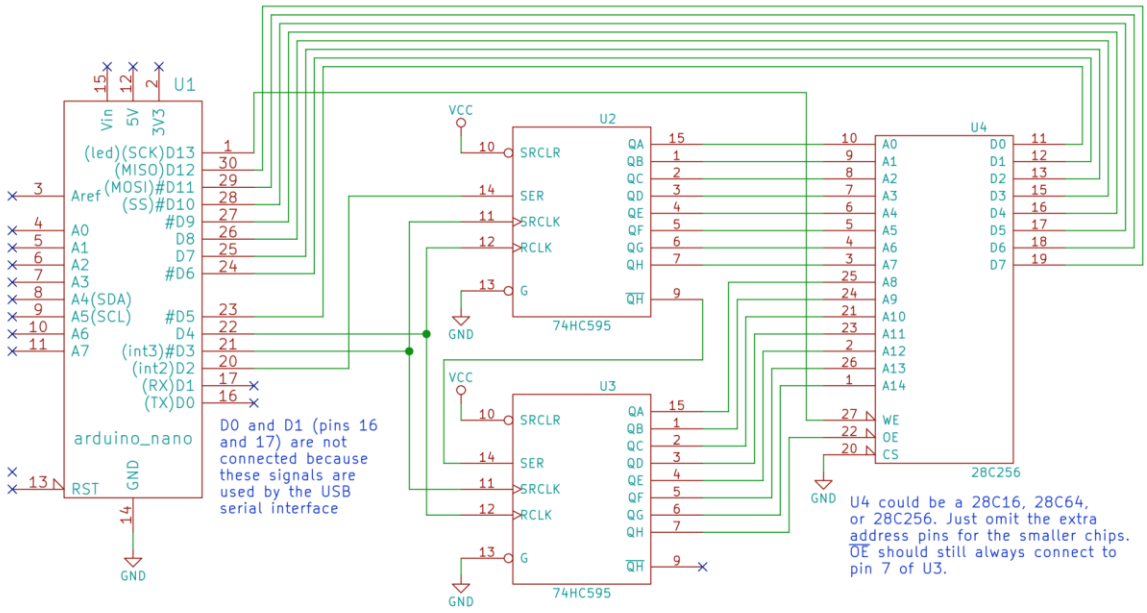


이와 같이 각 플립플롭의 출력이 또 다른 플립플롭의 클럭 신호로 연결되어 차례로 플립플롭이 활성화된다는 사실을 확인할 수 있다.

5. 기타 이론

시프트 레지스터는 보통 범용 입출력 핀이 부족한 경우 한 번에 여러 데이터를 전송하기 위해 사용된다. 예를 들어, 아두이노를 사용하여 EEPROM과 같은 저장 장치에 데이터를 쓸 때 사용한다. 이러한 저장장치는 병렬적으로 작동하기 때문에 한 번에 최소 8비트의 데이터를 입력해야 할 필요가 있는데, 아두이노와 같은 마이크로컨트롤러를 사용하여 저장 장치를 플래싱하기에는 범용 입출력 핀(GPIO)이 부족한 경우가 많다.

이러한 불편함을 해소하기 위해 시프트 레지스터가 자주 사용된다. 다음과 같은 회로도가 있다고 하자.



여기서 74HC595 IC는 8비트 시프트 레지스터이고, 28C256은 최대 256K의 정보를 저장할 수 있는 EEPROM이다. 28C256 IC는 256K의 정보에 접근하기 위해 15개의 주소 입력이 필요한데, 15개의 주소 입력과 8비트 데이터를 한 번에 입력할 수 있을 정도로 아두이노의 GPIO의 개수가 많지 않다. 이를 해결하기 위해 시프트 레지스터를 사용할 수 있다. 다음 코드를 사용하면 시프트 레지스터에 미리 접근할 주소를 모두 저장한 후, 28C256의 WE(Write Enable) 입력을 설정하여 한 번에 어떤 주소에 1바이트의 데이터를 저장할 수 있다.

```
#define SHIFT_DATA 2
#define SHIFT_CLK 3
#define SHIFT_LATCH 4
#define EEPROM_D0 5
#define EEPROM_D7 12
#define WRITE_EN 13

/*
 * Output the address bits and outputEnable signal using shift
registers.
 */
void setAddress(int address, bool outputEnable)
{
    shiftOut(SHIFT_DATA, SHIFT_CLK, MSBFIRST, (address >> 8) |
(outputEnable ? 0x00 : 0x80));
    shiftOut(SHIFT_DATA, SHIFT_CLK, MSBFIRST, address);
```

```
    digitalWrite(SHIFT_LATCH, LOW);
    digitalWrite(SHIFT_LATCH, HIGH);
    digitalWrite(SHIFT_LATCH, LOW);
}

/*
 * Read a byte from the EEPROM at the specified address.
 */
byte readEEPROM(int address)
{
    for (int pin = EEPROM_D0; pin <= EEPROM_D7; pin += 1) {
        pinMode(pin, INPUT);
    }
    setAddress(address, /*outputEnable*/ true);

    byte data = 0;
    for (int pin = EEPROM_D7; pin >= EEPROM_D0; pin -= 1) {
        data = (data << 1) + digitalRead(pin);
    }
    return data;
}

/*
 * Write a byte to the EEPROM at the specified address.
 */
void writeEEPROM(int address, byte data)
{
    setAddress(address, /*outputEnable*/ false);
    for (int pin = EEPROM_D0; pin <= EEPROM_D7; pin += 1) {
        pinMode(pin, OUTPUT);
    }

    for (int pin = EEPROM_D0; pin <= EEPROM_D7; pin += 1) {
        digitalWrite(pin, data & 1);
        data = data >> 1;
    }
    digitalWrite(WRITE_EN, LOW);
    delayMicroseconds(1);
    digitalWrite(WRITE_EN, HIGH);
    delay(10);
}

/*
 * Read the contents of the EEPROM and print them to the serial monitor.
 */
void printContents()
```

```
{
    for (int base = 0; base <= 255; base += 16) {
        byte data[16];
        for (int offset = 0; offset <= 15; offset += 1) {
            data[offset] = readEEPROM(base + offset);
        }

        char buf[80];
        sprintf(buf,
"%03x:  %02x %02x %02x %02x %02x %02x %02x %02x   %02x %02x %02x %02x %0
2x %02x %02x %02x",
            base, data[0], data[1], data[2], data[3], data[4], data[5],
data[6], data[7],
            data[8], data[9], data[10], data[11], data[12], data[13],
data[14], data[15]);

        Serial.println(buf);
    }
}

// 4-bit hex decoder for common anode 7-segment display
byte data[] = { 0x81, 0xcf, 0x92, 0x86, 0xcc, 0xa4, 0xa0, 0x8f, 0x80,
0x84, 0x88, 0xe0, 0xb1, 0xc2, 0xb0, 0xb8 };

// 4-bit hex decoder for common cathode 7-segment display
// byte data[] = { 0x7e, 0x30, 0x6d, 0x79, 0x33, 0x5b, 0x5f, 0x70, 0x7f,
0x7b, 0x77, 0x1f, 0x4e, 0x3d, 0x4f, 0x47 };

void setup()
{
    // put your setup code here, to run once:
    pinMode(SHIFT_DATA, OUTPUT);
    pinMode(SHIFT_CLK, OUTPUT);
    pinMode(SHIFT_LATCH, OUTPUT);
    digitalWrite(WRITE_EN, HIGH);
    pinMode(WRITE_EN, OUTPUT);
    Serial.begin(57600);

    // Erase entire EEPROM
    Serial.print("Erasing EEPROM");
    for (int address = 0; address <= 2047; address += 1) {
        writeEEPROM(address, 0xff);

        if (address % 64 == 0) {
            Serial.print(".");
        }
```

```
    }
    Serial.println(" done");

    // Program data bytes
    Serial.print("Programming EEPROM");
    for (int address = 0; address < sizeof(data); address += 1) {
        writeEEPROM(address, data[address]);

        if (address % 64 == 0) {
            Serial.print(".");
        }
    }
    Serial.println(" done");

    // Read and print out the contents of the EERPROM
    Serial.println("Reading EEPROM");
    printContents();
}
```