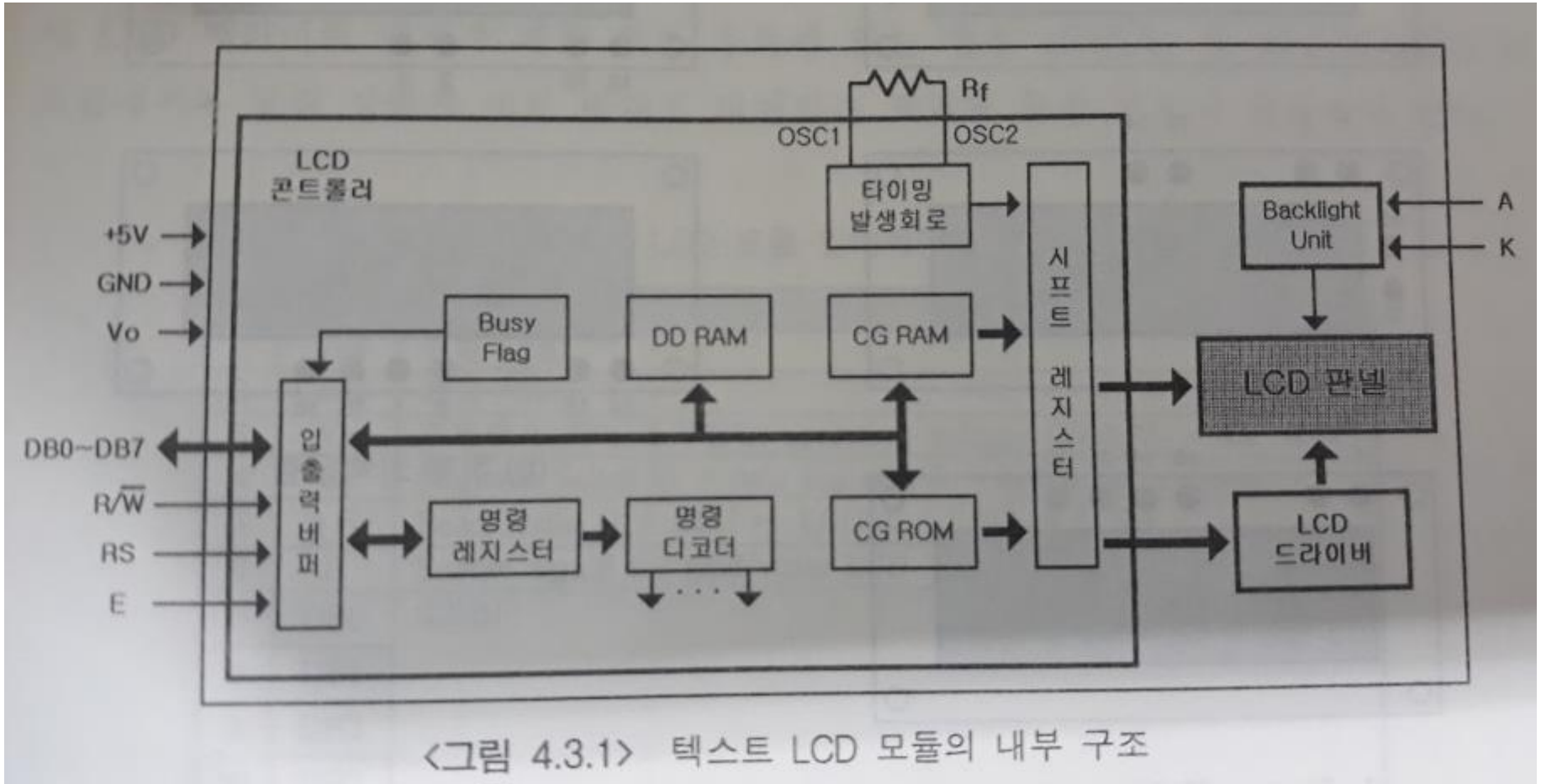
TEXT-LCD 인터페이스

2019.01.29
작성자 : 안상재
sangjae2015@naver.com

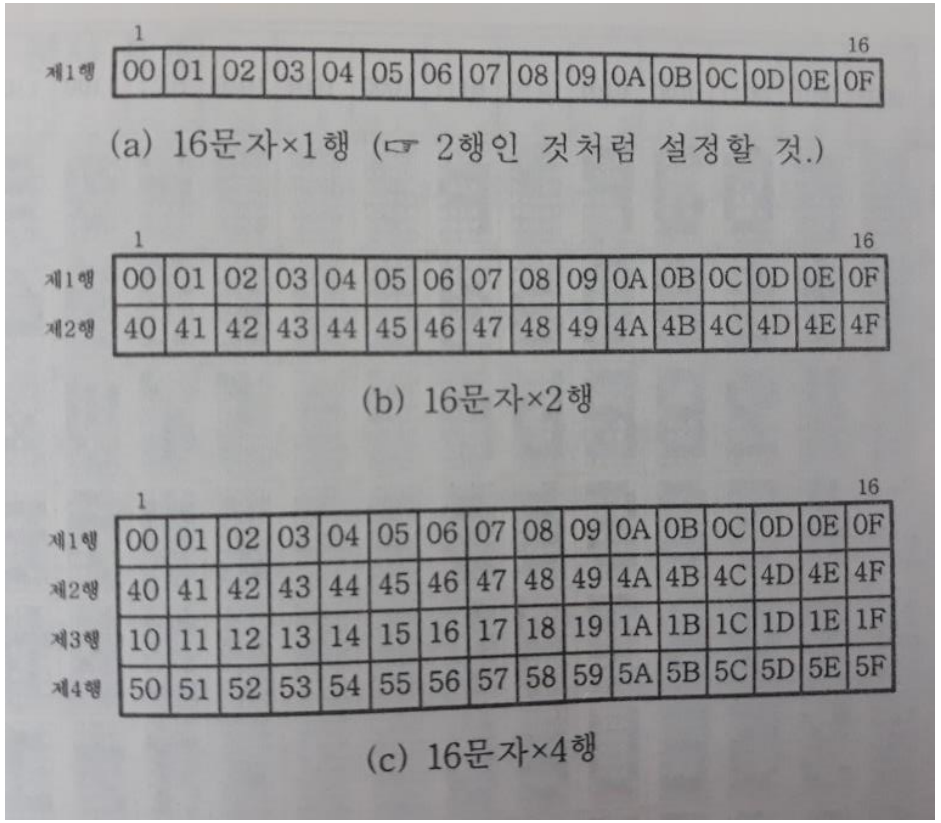
1. 내부구조



<그림 4.3.1> 텍스트 LCD 모듈의 내부 구조

2. 메모리

- **DD RAM :** 텍스트 LCD에 표시할 문자들의 ASCII 코드 데이터가 저장되는 내부 메모리이며 모두 80개의 번지가 있음.
LCD 화면의 각 행과 열의 문자 위치에는 고유한 어드레스 값이 부여되어 있다.



(a) 16문자×1행 (☞ 2행인 것처럼 설정할 것.)

	1															16
제1행	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F

(b) 16문자×2행

	1															16
제1행	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
제2행	40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F

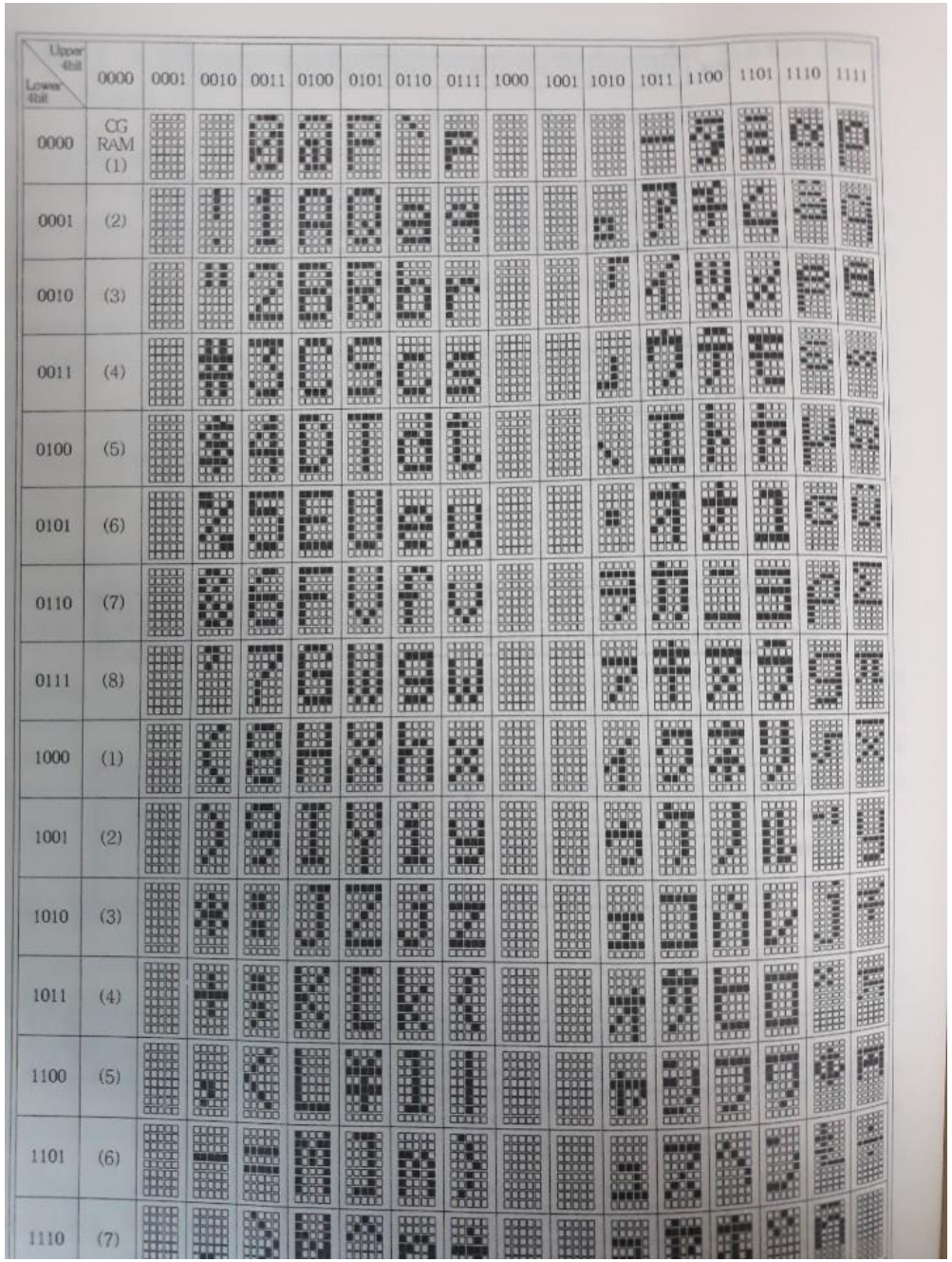
(c) 16문자×4행

	1															16
제1행	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
제2행	40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F
제3행	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
제4행	50	51	52	53	54	55	56	57	58	59	5A	5B	5C	5D	5E	5F

DDRAM 의 어드레스

- **CG ROM :** 192개(일본형) 또는 240개(유럽형)의 기본 문자 폰트가 저장되어 있고, 각 문자 폰트에 해당하는 문자 코드를 DD RAM에 써주기만 하면 이것에 해당하는 폰트가 자동으로 CG ROM에서 찾아져서 화면에 디스플레이 된다.

- **CG RAM :** CG ROM에 지정된 기본 문자 이외의 문자를 화면에 표시하려면 사용자 정의문자를 만들어 사용할 수 있고, 사용자 정의문자를 지정하는데 사용하는 메모리가 CG RAM 이다.

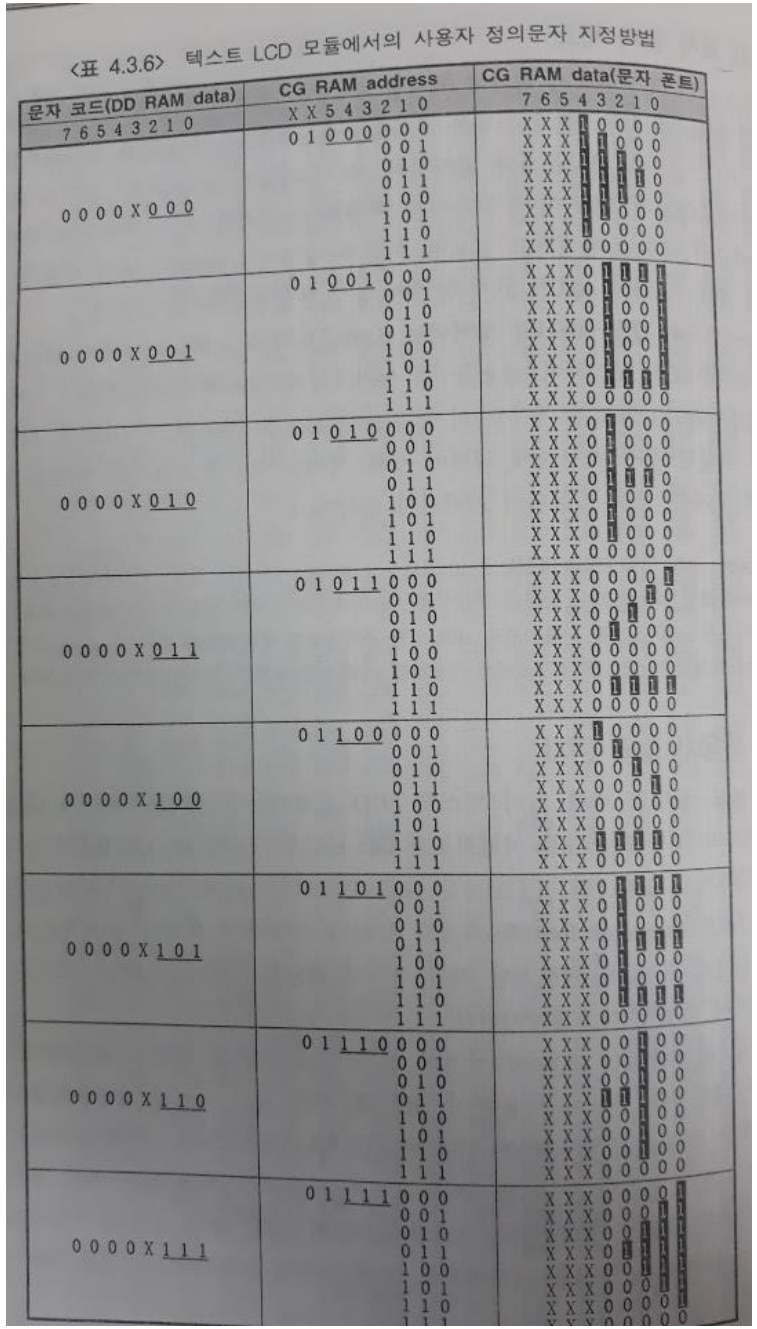


Upper 4bit / Lower 4bit	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
0000	CG RAM (1)															
0001	(2)															
0010	(3)															
0011	(4)															
0100	(5)															
0101	(6)															
0110	(7)															
0111	(8)															
1000	(1)															
1001	(2)															
1010	(3)															
1011	(4)															
1100	(5)															
1101	(6)															
1110	(7)															

CG ROM과 CG RAM

- CG ROM에 저장되어 있는 문자를 해당하는 문자코드를 DDRAM에 써주면 LCD 화면에 출력할 수 있음. 문자코드는 ASCII 코드 형태로 되어 있음

- 예를 들어, DD RAM의 00H번지에 'A' 라는 문자를 표시하려면
먼저, IR 레지스터에 명령어 0x80를 써주어서 DD RAM의 어드레스를 0X00으로 설정하고 DR 레지스터에 문자 'A'에 해당하는 0x41 을 써주면 된다.



〈표 4.3.6〉 텍스트 LCD 모듈에서의 사용자 정의문자 지정방법

문자 코드(DD RAM data)	CG RAM address	CG RAM data(문자 폰트)
7 6 5 4 3 2 1 0	X X 5 4 3 2 1 0	7 6 5 4 3 2 1 0
0 0 0 0 X 0 0 0	0 1 0 0 0 0 0	X X X 1 0 0 0 0
	0 0 1	X X X 1 1 0 0 0
	0 1 0	X X X 1 0 1 0 0
	0 1 1	X X X 1 1 1 1 0
	1 0 0	X X X 1 0 0 1 0
	1 0 1	X X X 1 0 0 0 0
	1 1 0	X X X 1 0 0 0 0
	1 1 1	X X X 0 0 0 0 0
0 0 0 0 X 0 0 1	0 1 0 0 1 0 0 0	X X X 0 1 1 1 1
	0 0 1	X X X 0 1 0 0 1
	0 1 0	X X X 0 1 0 0 1
	0 1 1	X X X 0 1 0 0 1
	1 0 0	X X X 0 1 0 0 1
	1 0 1	X X X 0 1 0 0 1
	1 1 0	X X X 0 1 1 1 1
	1 1 1	X X X 0 0 0 0 0
0 0 0 0 X 0 1 0	0 1 0 1 0 0 0	X X X 0 1 0 0 0
	0 0 1	X X X 0 1 0 0 0
	0 1 0	X X X 0 1 0 0 0
	0 1 1	X X X 0 1 1 1 0
	1 0 0	X X X 0 1 0 0 0
	1 0 1	X X X 0 1 0 0 0
	1 1 0	X X X 0 1 0 0 0
	1 1 1	X X X 0 0 0 0 0
0 0 0 0 X 0 1 1	0 1 0 1 1 0 0 0	X X X 0 0 0 0 1
	0 0 1	X X X 0 0 0 1 0
	0 1 0	X X X 0 0 1 0 0
	0 1 1	X X X 0 1 0 0 0
	1 0 0	X X X 0 0 0 0 0
	1 0 1	X X X 0 0 0 0 0
	1 1 0	X X X 0 1 1 1 1
	1 1 1	X X X 0 0 0 0 0
0 0 0 0 X 1 0 0	0 1 1 0 0 0 0	X X X 1 0 0 0 0
	0 0 1	X X X 0 1 0 0 0
	0 1 0	X X X 0 0 1 0 0
	0 1 1	X X X 0 0 0 1 0
	1 0 0	X X X 0 0 0 0 0
	1 0 1	X X X 0 0 0 0 0
	1 1 0	X X X 1 1 1 1 0
	1 1 1	X X X 0 0 0 0 0
0 0 0 0 X 1 0 1	0 1 1 1 0 1 0 0	X X X 0 1 1 1 1
	0 0 1	X X X 0 1 0 0 0
	0 1 0	X X X 0 1 0 0 0
	0 1 1	X X X 0 1 1 1 1
	1 0 0	X X X 0 1 0 0 1
	1 0 1	X X X 0 1 0 0 0
	1 1 0	X X X 0 1 1 1 1
	1 1 1	X X X 0 0 0 0 0
0 0 0 0 X 1 1 0	0 1 1 1 1 0 0 0	X X X 0 0 1 0 0
	0 0 1	X X X 0 0 0 1 0
	0 1 0	X X X 0 0 0 0 1
	0 1 1	X X X 0 0 0 1 0
	1 0 0	X X X 0 0 1 0 0
	1 0 1	X X X 0 1 0 0 0
	1 1 0	X X X 1 1 1 1 0
	1 1 1	X X X 0 0 0 0 0
0 0 0 0 X 1 1 1	0 1 1 1 1 1 0 0	X X X 0 0 0 0 0
	0 0 1	X X X 0 0 0 0 0
	0 1 0	X X X 0 0 0 0 0
	0 1 1	X X X 0 0 0 0 0
	1 0 0	X X X 0 0 0 0 0
	1 0 1	X X X 0 0 0 0 0
	1 1 0	X X X 0 0 0 0 0
	1 1 1	X X X 0 0 0 0 0

CG RAM에 사용자 정의문자 지정방법

- CG RAM 01000000 에서 비트 b6,b7는 0,1로 고정이다. 비트 b3~b5는 LCD에 표시되는 위치를 나타내는 비트이며, 비트 b0~b2는 한 문자를 표시하기 위한 각 행을 나타낸다.

- CG RAM에 쓰여지는 모든 문자는 5*8 도트이며 마지막 8번째 줄은 커서가 나타나는 위치이다. 이 CG RAM에는 5*7 도트를 사용하는 경우에는 최대 8문자까지 정의할 수 있고, 5*10 도트의 경우에는 최대 4문자까지 정의하여 사용할 수 있다.

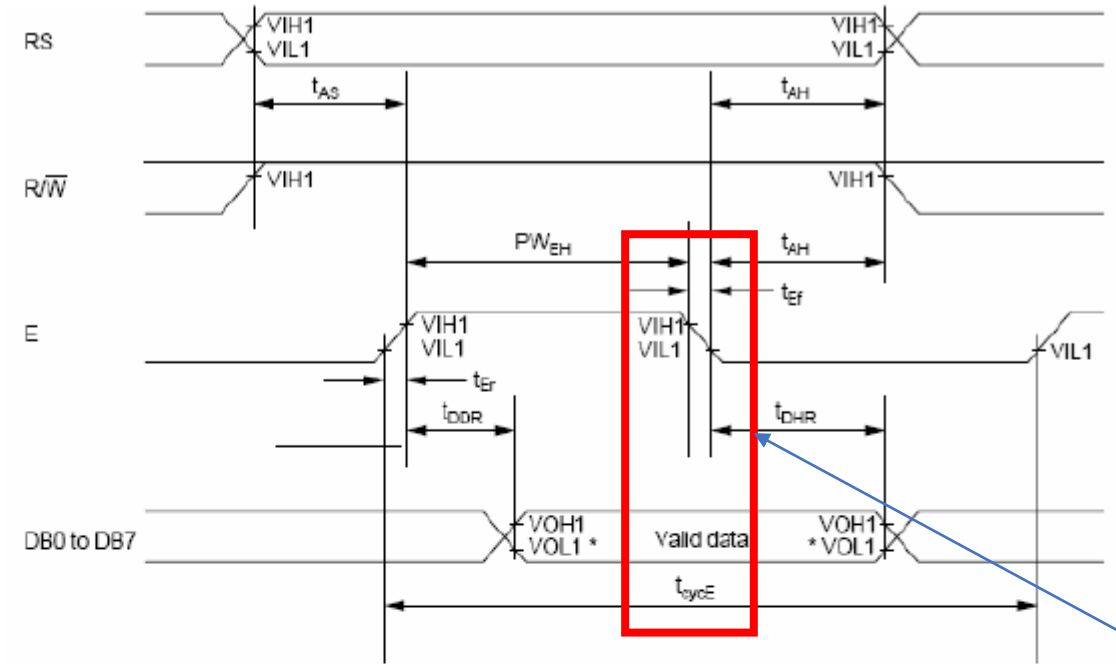
- 예를 들어, 0x40 을 명령어로 쓰고 화살표 문자에 해당하는 데이터 8바이트를 차례로 쓰면 DDRAM 0x00 번지에 화살표 문자가 저장이 된다.
  그 다음에 화살표 문자를 쓰고 싶을 때는 0x00 데이터를 쓰면 된다.

3. LCD 모듈의 단자

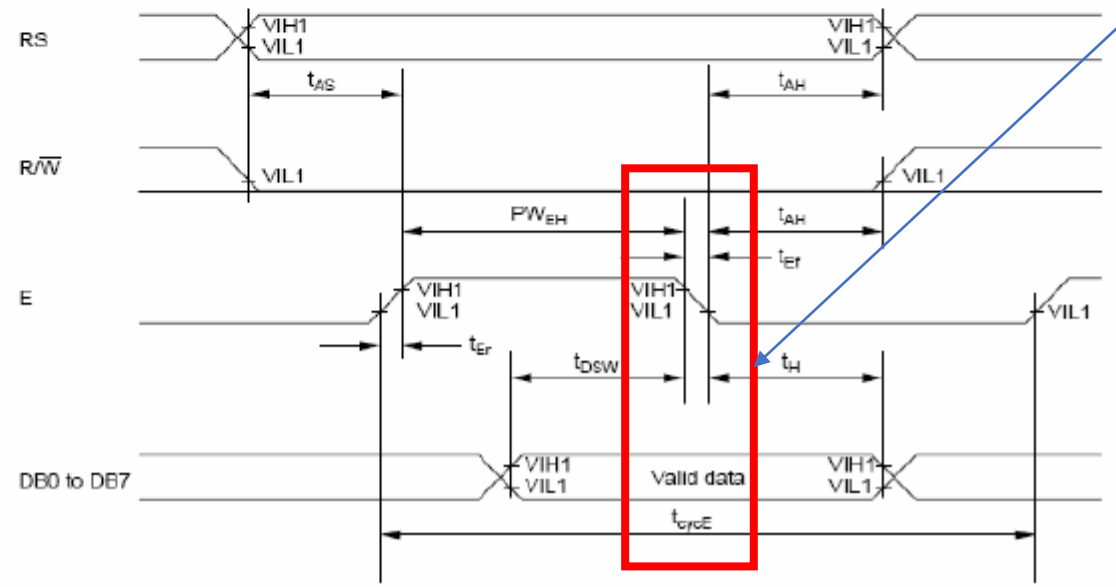
핀 번호	기 호	레 벨	기 능	
1	V_{SS}	-	0V	
2	V_{DD}	-	+5V	
3	Vo		LCD 밝기 조정(가변저항)	
4	RS	H/L	L : 명령 입력 (IR 선택) H : 데이터 입력 (DR 선택)	
5	$R/\overline{W}$	H/L	L : 쓰기(CPU → LCD module) H : 읽기(CPU ← LCD module)	
6	E	H	LCD 모듈의 허가 신호	
7	DB0	H/L	4비트 데이터 버스 이용시 사용 불가	8비트 데이터 버스 이용시 모두 사용
8	DB1	H/L		
9	DB2	H/L		
10	DB3	H/L		
11	DB4	H/L	4비트 데이터 버스 이용시 사용 가능	
12	DB5	H/L		
13	DB6	H/L		
14	DB7	H/L		

4. READ/WRITE 타이밍도

읽기 동작 타이밍도



쓰기 동작 타이밍도



E의 low edge에서 DB0 to DB7의 데이터가 READ 또는 WRITE 됨.

5. LCD 컨트롤러의 명령

각 핀의 신호를 명령표와 타이밍도에 맞게 설정해서 데이터를 전송해주면 됨. (GPIO)

명령	코드 RS	R/W	DB_7	DB_6	DB_5	DB_4	DB_3	DB_2	DB_1	DB_0	기 능	실행 시간
화면 지움	0	0	0	0	0	0	0	0	0	1	화면을 클리어하고, 커서가 홈 위치인 0번지로 돌아간다.	1.64ms
커서 홈	0	0	0	0	0	0	0	0	1	×	커서를 홈 위치로 돌아가게 한다. 또한 시프트되어 표시된 것도 되돌아가게 된다. DDRAM의 내용은 변하지 않는다.	1.64ms
엔트리 모드세트	0	0	0	0	0	0	0	1	I/D	S	데이터를 쓰거나 읽기를 수행할 때의 동작 모드를 결정한다. 즉, 커서의 진행 방향과 화면을 자동으로 시프트 시킬 것인지를 결정한다.	40㎲
화면 ON/OFF	0	0	0	0	0	0	1	D	C	B	화면 표시 ON/OFF(D), 커서 ON/ OFF(C), 커서 위치에 있는 문자의 블링크 기능(B)을 설정한다.	40㎲
커서/표시 시프트	0	0	0	0	0	1	S/C	R/L	×		화면 표시 내용은 변경시키지 않고, 커서와 화면의 이동과 시프트 동작을 설정한다.	40㎲
기능 설정	0	0	0	0	1	DL	N	F	×	×	LCD의 인터페이스 데이터 길이(DL), 표시 행수(N), 문자 폰트(F) 등을 설정한다.	40㎲
CGRAM 주소 설정	0	0	0	1	A_{CG}						CGRAM의 주소를 설정한다. 이후 전송되는 데이터는 CGRAM의 데이터이다.	40㎲
DDRAM 주소 설정	0	0	1	A_{DD}							DDRAM의 주소를 설정한다. 이후 전송되는 데이터는 DDRAM의 데이터이다.	40㎲
BF/주소 설정	0	1	BF	AC							LCD 모듈의 동작 여부와 현재 설정된 주소의 내용을 알기 위해서 BF 및 AC의 내용을 읽는다. CGRAM, DDRAM 양쪽 모두 사용할 수 있다.	0㎲
CG RAM, DD RAM으로 데이터 써넣기	1	0	써넣을 데이터								DDRAM 또는 CGRAM에 데이터를 써넣는다.	40㎲
CG RAM, DD RAM에서 데이터 읽기	1	1	읽을 데이터								DDRAM 또는 CGRAM에서 데이터를 읽는다.	40㎲

A_{CG} : CGRAM 주소
A_{DD} : DDRAM 주소
AC : 주소 카운터

I/D=1 : 증가(+1)
I/D=0 : 감소(−1)
S=1 : 표시 시프트 ON
S=0 : 표시 시프트 OFF

D=1 : 화면 ON
C=1 : 커서 ON
B=1 : 블링크 ON

D=0 : 화면 OFF
C=0 : 커서 OFF
B=0 : 블링크 OFF

R/L=1 : 우 시프트
S/C=1 : 화면 이동

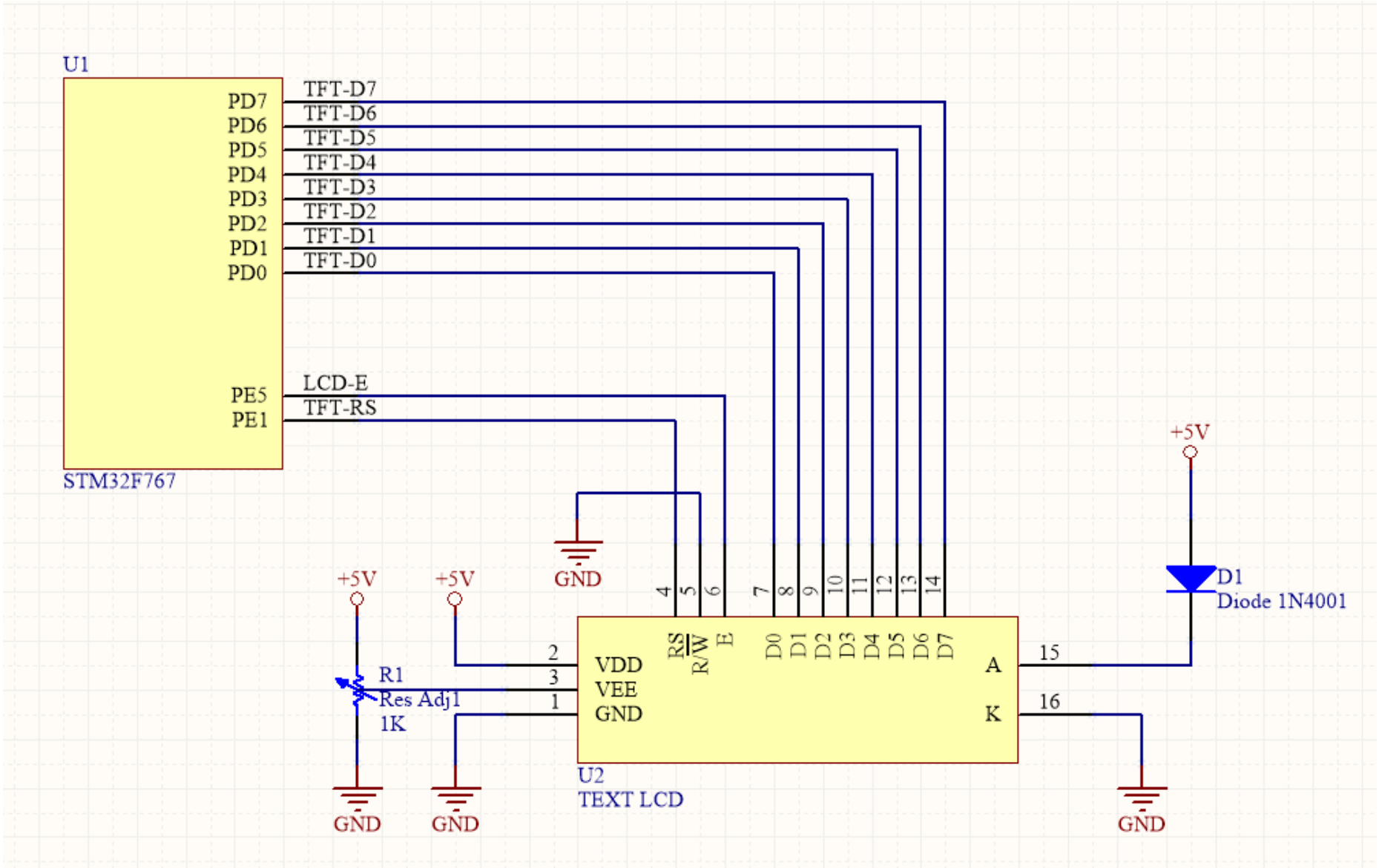
R/L=0 : 좌 시프트
S/C=0 : 커서 이동

DL=1 : 8비트
N=1 : 2행
F=1 : 4*10도트

DL=0 : 4비트
N=0 : 1행
F=0 : 5*7도트

BF=1 : 내부 동작중 BF=0 : 명령 쓰기 가능

6. MCU-LCD 회로도


U1
PD7
PD6
PD5
PD4
PD3
PD2
PD1
PD0
TFT-D7
TFT-D6
TFT-D5
TFT-D4
TFT-D3
TFT-D2
TFT-D1
TFT-D0
PE5
PE1
LCD-E
TFT-RS
STM32F767
+5V
+5V
GND
R1
Res Adj1
1K
VDD
VEE
GND
RS
R/W
E
D0
D1
D2
D3
D4
D5
D6
D7
A
K
U2
TEXT LCD
+5V
D1
Diode 1N4001
GND
GND
GND

7. 소스 코드

7-1. LCD 모듈 관련 함수

*** 회로도에서 LCD의 R/W(5번 핀)은 GND에 연결시켰기 때문에 따로 신호를 인가하지 않아도 됨!**

```
void Initialize_LCD(void)  /* LCD 초기화 */
{
  RCC->AHB1ENR |= 0x00000018;  // GPIOE,GPIOD Clock Enable

  GPIOD->MODER &= 0xFFFF0000;
  GPIOD->MODER |= 0x00005555;  // PD0~7 출력 설정
  GPIOD->ODR &= 0xFFFF0000;  // PD0~7 = 0
  GPIOD->OSPEEDR &= 0xFFFF0000;
  GPIOD->OSPEEDR |= 0x00005555;

  GPIOE->MODER &= 0xFFFFFF3F3;
  GPIOE->MODER |= 0x00000404;  // PE1,5 출력 설정
  GPIOE->ODR &= 0xFFFFFFDD;  // PE1,5 = 0
  GPIOE->OSPEEDR &= 0xFFFFF3F3;
  GPIOE->OSPEEDR |= 0x00000404;

  LCD_command(0x38);  // function set (8 bit, 2 line, 5*7dot)
  LCD_command(0x0C);  // display control (display on, cursor off)
  LCD_command(0x06);  // entry mode set (cursor right increment, display not shift)
  LCD_command(0x01); // clear display
  Delay_ms(2);
}
```

```
void LCD_command(U08 command) /* write command */
{
  GPIOE->BSRR = 0x00220000; // E=0, RS=0 (command mode)
  GPIOD->ODR = command; // output command
  Delay_us(1);
  GPIOE->BSRR = 0x00000020; // E=1
  Delay_us(1);
  GPIOE->BSRR = 0x00200000; // E=0
  Delay_us(50);
}
```

```
void LCD_data(U08 data) /* write data */
{
  GPIOE->BSRR = 0x00200002; // E=0, RS=1 (data mode)
  GPIOE->ODR = data; //output data
  Delay_us(1);
  GPIOE->BSRR = 0x00000020; // E=1
  Delay_us(1);
  GPIOE->BSRR = 0x00200000; // E=0
  Delay_us(50);
}
```

```
void LCD_string(U08 command, U08 *string) /* display a string on LCD */
{
  LCD_command(command);
  while(*string != '₩n0')
  {
    LCD_data(*string);
    string++;
  }
}
```

7-2. 수치 데이터 출력 함수

```
void LCD_2d(unsigned int number)  /* display 2-digit decimal number */
{
  unsigned int i;
  i = number/10;

  if(i == 0)
    LCD_data(' ');
  else
    LCD_data(i + '0');

  i = number % 10;
  LCD_data(i+'0');
}
```

```
void LCD_3d(unsigned int number) /* display 3-digit decimal number */
{
  unsigned int i, flag;

  flag = 0;
  i = number/100;

  if(i == 0)
    LCD_data(' ');
  else
  {
    LCD_data(i + '0');
    flag = 1;
  }

  number = number % 100;
  i = number/10;
  if((i==0) && (flag == 0))
    LCD_data(' ');
  else
  {
    LCD_data(i + '0');
    flag = 1;
  }

  i = number % 10;
  LCD_data(i + '0');
}
```

```
void LCD_4d(unsigned int number)  /* display 4-digit decimal number */
{
  unsigned int i, flag;

  flag = 0;
  i = number/1000;

  if(i == 0)
    LCD_data(' ');
  else
  {
    LCD_data(i + '0');
    flag = 1;
  }

  number = number % 1000;
  i = number/100;
  if((i==0) && (flag == 0))
    LCD_data(' ');
  else
  {
    LCD_data(i + '0');
    flag = 1;
  }

  i = numer % 10;
  LCD_data(i + '0');
}
```

```
void LCD_2hex(unsigned int number)  /* display 2-digit hex number (bin -> hex) */
{
  unsigned int i;

  i = (number >> 4) & 0x0F;
  if(i <= 9)
    LCD_data(i + '0');
  else
    LCD_data(i – 10 + 'A');

  i = number & 0x0F;
  if(i <= 9)
    LCD_data(i + '0');
  else
    LCD_data(i - 10 + 'A');
}
```

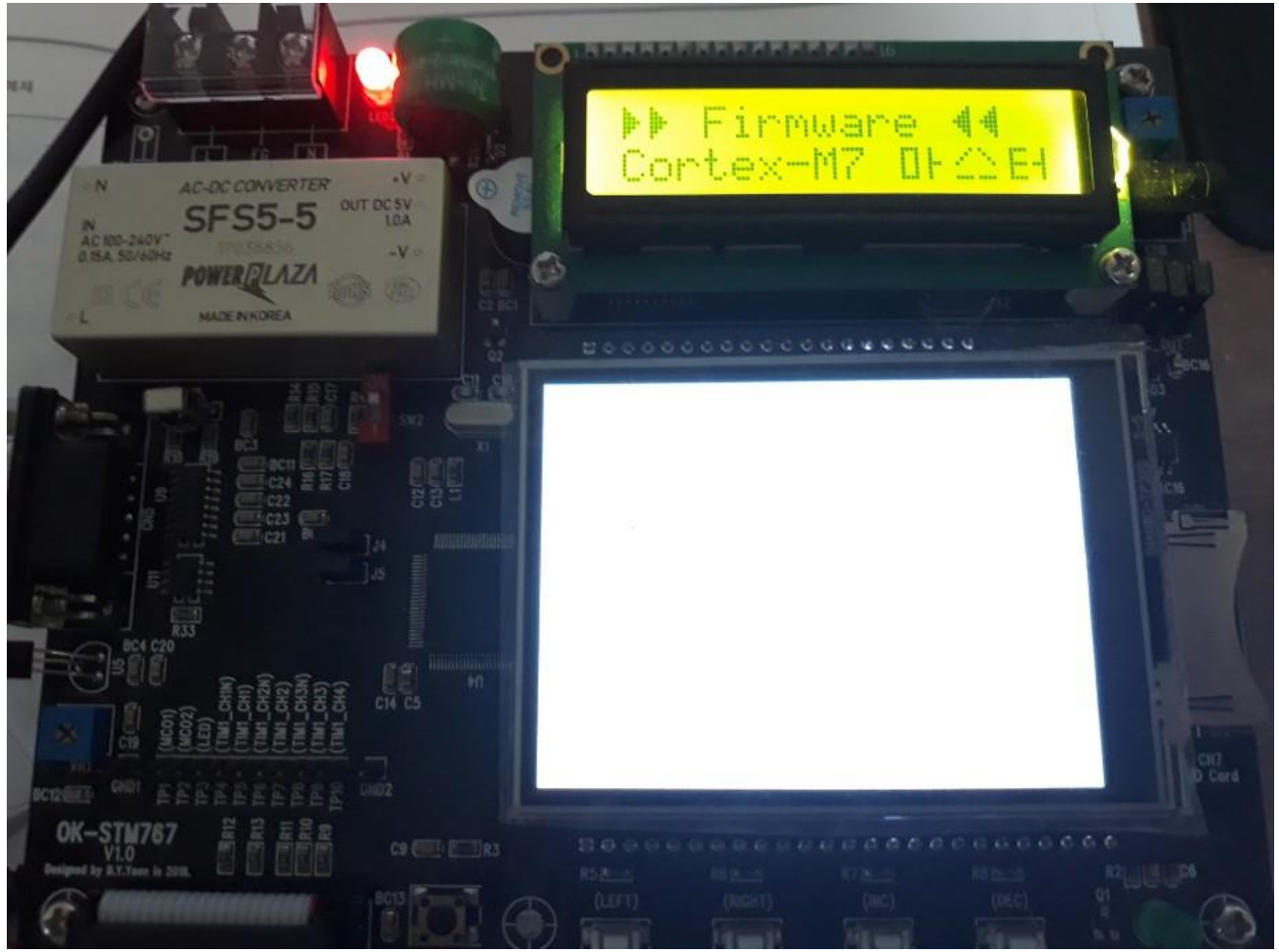
```
void LCD_8bin(unsigned int number) /* display 8-bit binary number */
{
  LCD_data(((number >> 7) & 0x01) + '0');
  LCD_data(((number >> 6) & 0x01) + '0');
  LCD_data(((number >> 5) & 0x01) + '0');
  LCD_data(((number >> 4) & 0x01) + '0');
  LCD_data(((number >> 3) & 0x01) + '0');
  LCD_data(((number >> 2) & 0x01) + '0');
  LCD_data(((number >> 1) & 0x01) + '0');
  LCD_data((number & 0x01) + '0');
}
```

```
void LCD_hexadecimal(U32 number, U08 digit)  /* display hexadecimal number */
{
  unsigned char i, character;

  if((digit == 0) || (digit > 8)) return;

  for(i=digit; i>0; i--)
  {
    character = (number >> 4*(i-1)) & 0x0F;
    if(character < 10) LCD_data(character + '0');
    else  LCD_data(character – 10 + 'A');
  }
}


void LCD_0x_hexadecimal(U32 number, U08 digit)  /* display hexadecimal number with 0x */
{
  unsigned char i, character;

  if((digit == 0) || (digit > 8)) return;

  LCD_data('0');
  LCD_data('x');

  for(i=digit; i>0; i--)
  {
    character = (number >> 4*(i-1)) & 0x0F;
    if(character < 10) LCD_data(character + '0');
    else  LCD_data(character – 10 + 'A');
  }
}
```

```
/* display unsigned floating-point number */
void LCD_unsigned_float(float number, U08 integral, U08 fractional)
{
  unsigned char zero_flag, digit, character;
  unsigned long div, integer;

  digit = integral + fractional;
  if((integral == 0) || (fractional == 0) || (digit > 9))
    return;

  div = 1;
  while(--digit)
    div *= 10;

  while(fractional--)
    number *= 10.;
  integral = (U32) (number + 0.5);

  zero_flag = 0;
  digit = 1;
  while(div > 0)
  {
    character = integral / div;
    if((character == 0) && (zero_flag == 0) && (digit != integral))
      LCD_data(character + ' ');
    else
    {
      zero_flag = 1;
      LCD_data(character + '0');
    }
    integer %= div;
    div /= 10;

    if(digit == integral)
      LCD_data('.');
    digit++;
  }
}
```

```
/* display signed floating-point number */
void LCD_signed_float(float number, U08 integral, U08 fractional)
{
  unsigned char zero_flag, digit, character;
  unsigned long div, integer;

  digit = integral + fractional;
  if((integral == 0) || (fractional == 0) || (digit > 9)) return;

  if(number >= 0)
    LCD_data('+');
  else
  {
    LCD_data('-');
    number = -number;
  }

  div = 1;
  while(--digit)  div *= 10.;
  integer = (U32) (number + 0.5);

  zero_flag = 0;
  digit = 1;
  while(div > 0)
  {
    character = integral / div;
    if((character == 0) && (zero_flag == 0) && (digit != integral))
      LCD_data(character + ' ');
    else
    {
      zero_flag = 1;
      LCD_data(character + '0');
    }
    integral %= div;
    div /= 10;

    if(digit == integral)
      LCD_data('.');
    digit++;
  }
}
```

8. 결과 화면



OK-STM767 키트에 의한 TEXT-LCD 실습