마프2 작품계획서

- 가속도 센서와 GLCD를 이용한 레이싱 게임 -

날짜 : 2019 04 24

이름 : 강승우

학번 : 2014161001

1. 작품명

가속도 센서와 GLCD를 이용한 레이싱 게임

2. 작품 개요 및 동작

실제 핸들을 다루는 것처럼 손잡이를 돌려 조향할 수 있는 레이싱 게임입니다. 가속도 센서를 이용해서 핸들의 기울어진 정도를 읽어오고, 이를 바탕으로 GLCD(주 디스플레이)를 부착한 서보 모터를 회전시켜 화면의 평형을 유지합니다.

레이싱 게임의 가속과 감속은 아날로그 감압 센서를 이용해서 세밀하게 조정 가능하게끔 합니다.

실제 게임은 X, Y 평면상에서 이루어지며, 정점으로만 구성된 벡터 그래픽을 사용합니다. 거리에 따라 각 정점의 오프셋을 스케일링함으로써 원근감을 구현하게 됩니다.

3. 사용 포트 및 부품

<사용 부품 리스트>

부품명	규격	수량	기능
KUT-128	Pin Header	1	메인 MCU 보드
GLCD	LG2401283, 240x128	1	주 디스플레이
서보 모터	HES-288	1	화면 평형 유지
가속도 센서	ADXL202JQC, SMD	1	기울기 감지
SRAM	IS62C256AL-45ULI	1	추가 기억장치, 32KB
스피커	FQ-031	2	SFX 출력
압력 센서	FSR, RA12P	2	엑셀, 브레이크 아날로그 입력
8비트 D래치	74573	1	SRAM 연결용
정전압 레귤레이터	LM3940	1	LCD 바이어스용

<사용 포트> : 핀 단위

사용 포트	특수 기능	입/출력	연결 부품(핀)	기능
PA0	AD0	입출력	74573 D0	SRAM DATA0/ADDR0
PA1	AD1	입출력	74573 D1	SRAM DATA1/ADDR1
PA2	AD2	입출력	74573 D2	SRAM DATA2/ADDR2
PA3	AD3	입출력	74573 D3	SRAM DATA3/ADDR3
PA4	AD4	입출력	74573 D4	SRAM DATA4/ADDR4
PA5	AD5	입출력	74573 D5	SRAM DATA5/ADDR5
PA6	AD6	입출력	74573 D6	SRAM DATA6/ADDR6
PA7	AD7	입출력	74573 D7	SRAM DATA7/ADDR7
PC0	A8	출력	IS62C256AL AD8	SRAM ADDR8
PC1	A9	출력	IS62C256AL AD9	SRAM ADDR9
PC2	A10	출력	IS62C256AL AD10	SRAM ADDR10
PC3	A11	출력	IS62C256AL AD11	SRAM ADDR11
PC4	A12	출력	IS62C256AL AD12	SRAM ADDR12
PC5	A13	출력	IS62C256AL AD13	SRAM ADDR13
PC6	A14	출력	IS62C256AL AD14	SRAM ADDR14
PG0	/WR	출력	IS62C256AL /WE	SRAM WRITE EN
PG1	/RD	출력	IS62C256AL /OE	SRAM OUTPUT EN
PG2	ALE	출력	74573 LE	데이터 / 어드레스 전환
PD0		출력	LG2401283 D0	GLCD D0/D4
PD1		출력	LG2401283 D1	GLCD D1/D5
PD2		출력	LG2401283 D2	GLCD D2/D6
PD3		출력	LG2401283 D3	GLCD D3/D7
PD4		출력	LG2401283 WR0	GLCD WRITE CLK
PD5		출력	LG2401283 CD	GLCD I/D SELECT
PF0	ADC0	입력	FSR 0번	아날로그 압력 입력 0
PF1	ADC1	입력	FSR 1번	아날로그 압력 입력 1

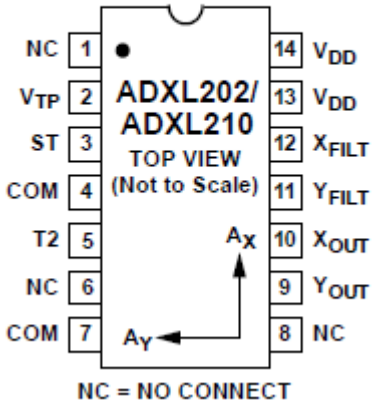
PF2		입력	ADXL 202 X_{OUT}	가속도 X축 PWM 입력
PF3		입력	ADXL 202 Y_{OUT}	가속도 Y축 PWM 입력
PB4	OC0	출력	SPK 0	0번 스피커 주파수 변조 출력
PB7	OC2	출력	SPK 1	1번 스피커 주파수 변조 출력
PB6		출력	HES-288	서보모터 PWM 제어신호
PE0		입력	출력 커넥터	스위치 버튼 0
PE1		입력	출력 커넥터	스위치 버튼 1
PE2		입력	출력 커넥터	스위치 버튼 2
PE3		입력	출력 커넥터	스위치 버튼 3
PE4		입력	출력 커넥터	스위치 버튼 4
PE5		입력	출력 커넥터	스위치 버튼 5
PE6		입력	출력 커넥터	스위치 버튼 6
PE7		NC		

<사용 부품 사양서>

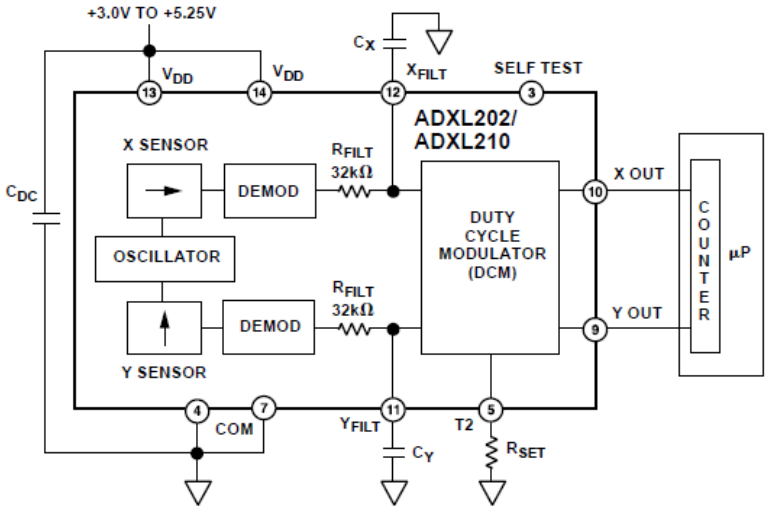
ADXL202 – 디지털 출력 가속도 감지 센서

PIN	NAME	DESCRIPTION
1	NC	No Connect
2	V_{TP}	Test Point, do not connect
3	ST	Self Test
4	COM	Common
5	T_2	Connect R_{SET} to Set T2 Period
6	NC	
7	COM	
8	NC	
9	Y_{OUT}	Y Axis duty cycle output
10	X_{OUT}	X Axis duty cycle output
11	Y_{FILT}	Connect capacitor for Y filter
12	X_{FILT}	Connect capacitor for X filter
13	V_{DD}	+3V to +5.25V, Connect to 14
14	V_{DD}	+3V to +5.25V, Connect to 13

PIN CONFIGURATION



T1과 T2포트의 Duty Ratio를 통해 아날로그 측정값을 디지털로 출력하는 가속도 센서이다. 일반적인 입력 포트에 연결한 뒤, 카운터로 일정 주기 동안의 1의 개수를 세는 방식으로 Duty Ratio를 역산할 수 있다.



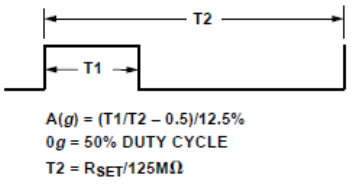


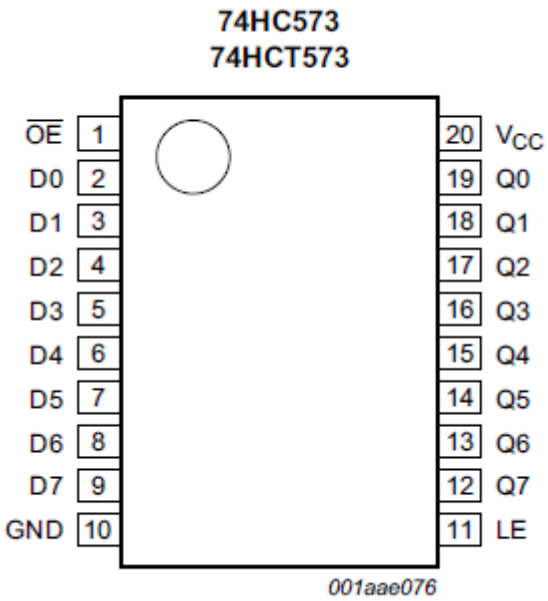
Figure 13. Block Diagram

주기 T2와 필터 대역폭을 산정하기 위한 저항과 캐패시터 값은 아래와 같다.

T2	R_{SET}
1 ms	125 kOhm
2 ms	250 kOhm
5 ms	625 kOhm
10 ms	1.25 MOhm

Bandwidth	Capacitor Value
10 Hz	0.47 uF
50 Hz	0.10 uF
100 Hz	0.05 uF
200 Hz	0.027 uF
500 Hz	0.01 uF
5 kHz	0.001 uF

74573 IC – 8BIT D LATCH



일반적인 8비트 Latch IC이다. ATMEGA 128의 PORT A를 이용하는 외부 램 인터페이스가 하위 8비트의 데이터와 어드레스를 공유하기 때문에, 먼저 어드레스를 LATCH한 상태에서 데이터를 읽거나 써야 한다.

이 작업은 하드웨어 내부에서 자동으로 이루어지므로, 핀 할당과 내부 레지스터 설정만 올바르게 했다면 더 신경 쓸 필요는 없다.

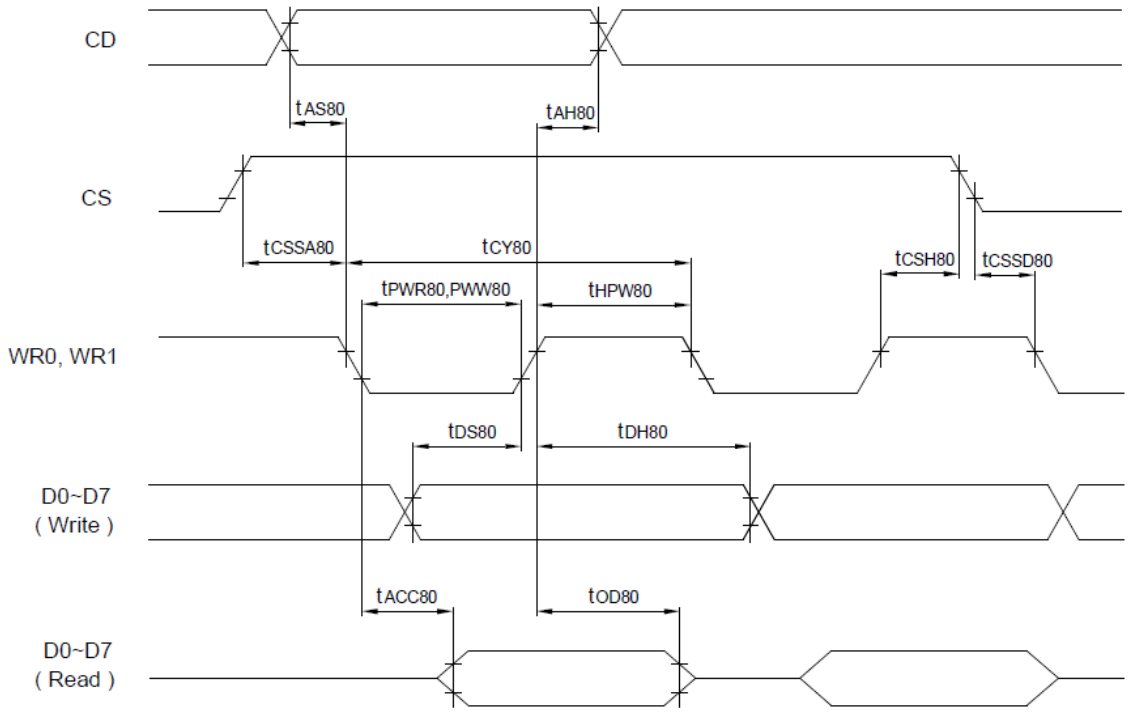
1.3 Terminal Functions

<table>
<tr><th>Pin No.</th><th>Symbol</th><th>Level</th><th>Function</th></tr>
<tr><td>1</td><td>VB1-</td><td>-</td><td rowspan="2">LCD Bias Voltages. These voltages are generated internally. Connect a 4.7uF/6.3V capacitor between VB1+ and VB1–.</td></tr>
<tr><td>2</td><td>VB1+</td><td>-</td></tr>
<tr><td>3</td><td>VB0-</td><td>-</td><td rowspan="2">LCD Bias Voltages. These voltages are generated internally. Connect a 4.7uF/6.3V capacitor between VB0+ and VB0–.</td></tr>
<tr><td>4</td><td>VB0+</td><td>-</td></tr>
<tr><td>5</td><td>VLCD</td><td>-</td><td>LCD driving voltage (VLCD is generated internally by UC1608). Connect a 0.1uF/25V capacitor and a 10MΩ resistor to VSS.</td></tr>
<tr><td>6</td><td>VBIAS</td><td></td><td>The reference voltage to generate LCD driving voltage. VBIAS can be used to fine turn VLCD (contrast) by external variable resistors. When use the internal resistor network, connect a 0.1uF capacitor to VSS.</td></tr>
<tr><td>7</td><td>VSS</td><td>0V</td><td>Ground</td></tr>
<tr><td>8</td><td>VDD</td><td>2.7 to 3.3V</td><td>Power supply for logic and charge pump</td></tr>
<tr><td>9</td><td>D7</td><td rowspan="8">H/L</td><td rowspan="8">Bi-directional bus for both serial and parallel host interfaces.
In serial modes, connect D0 to SCK, D3 to SDA.
<table>
<tr><td></td><td>BM[1:0]=1x
8-bit parallel</td><td>BM[1:0]=0x
4-bit parallel</td><td>BM[1:0]=01
S9</td><td>BM[1:0]=00
S8/S8uc</td></tr>
<tr><td>D0</td><td>D0</td><td>D0/D4</td><td>SCK</td><td>SCK</td></tr>
<tr><td>D1</td><td>D1</td><td>D1/D5</td><td>–</td><td>–</td></tr>
<tr><td>D2</td><td>D2</td><td>D2/D6</td><td>–</td><td>–</td></tr>
<tr><td>D3</td><td>D3</td><td>D3/D7</td><td>SDA</td><td>SDA</td></tr>
<tr><td>D4</td><td>D4</td><td>–</td><td>–</td><td>–</td></tr>
<tr><td>D5</td><td>D5</td><td>–</td><td>–</td><td>–</td></tr>
<tr><td>D6</td><td>D6</td><td>–</td><td>S9</td><td>S8/S8uc</td></tr>
<tr><td>D7</td><td>D7</td><td>0</td><td>1</td><td>1</td></tr>
</table>
Connect unused pins to VDD or VSS.</td></tr>
<tr><td>10</td><td>D6</td></tr>
<tr><td>11</td><td>D5</td></tr>
<tr><td>12</td><td>D4</td></tr>
<tr><td>13</td><td>D3</td></tr>
<tr><td>14</td><td>D2</td></tr>
<tr><td>15</td><td>D1</td></tr>
<tr><td>16</td><td>D0</td></tr>
<tr><td>17</td><td>WR1</td><td rowspan="2">H/L</td><td rowspan="2">WR[1:0] control the read/write operation of the host interface.
In 8080 mode: WR0 is /WR signal, WR1 is /RD signal.
In 6800 mode: WR0 is R/W signal, WR1 is Enable signal.
In serial modes: These two pins are not used, connect them to VSS.</td></tr>
<tr><td>18</td><td>WR0</td></tr>
<tr><td>19</td><td>CD</td><td>H/L</td><td>Data or instruction selection
L: D0 to D7 are Instruction code H: D0 to D7 are display data
In S9 mode, CD pin is not used and connect it to VDD or VSS.</td></tr>
<tr><td>20</td><td>/RST</td><td>L</td><td>Reset signal, active “L”. There is built-in power-on-reset circuit in UC1608. Connect /RST to VDD when it is not used.</td></tr>
<tr><td>21</td><td>CS</td><td>H</td><td>Chip selection signal, active “H”.</td></tr>
<tr><td>22</td><td>BM0</td><td rowspan="2">H/L</td><td rowspan="2">Bus mode selection. The interface bus mode is determined by BM[1:0] and [D7:D6] by the following relationship.
<table>
<tr><th>BM[1:0]</th><th>[D7:D6]</th><th>Mode</th></tr>
<tr><td>11</td><td>Data</td><td>6800/8-bit</td></tr>
<tr><td>10</td><td>Data</td><td>8080/8-bit</td></tr>
<tr><td>01</td><td>0x</td><td>6800/4-bit</td></tr>
<tr><td>00</td><td>0x</td><td>8080/4-bit</td></tr>
<tr><td>01</td><td>10</td><td>3-wire SPI w/ 9-bit token (S9: conventional)</td></tr>
<tr><td>00</td><td>10</td><td>4-wire SPI w/ 8-bit token (S8: conventional)</td></tr>
<tr><td>00</td><td>11</td><td>3/4-wire SPI w/ 8-bit token (S8uc: Ultra-Compact)</td></tr>
</table></td></tr>
<tr><td>23</td><td>BM1</td></tr>
</table>

240 * 128 해상도의 단색 그래픽 LCD이다. 특히, V_{DD} 입력 전압이 2.7~ 3.3V임에 유의해야 한다. 다이오드 등을 이용해 감압 후 바이어스 할 것...

3.2 Parallel Bus Timing Characteristics (8080 Series MPU, VDD=2.7V to 3.3V, Ta=25°C)

Description	Signal	Symbol	Condition	Min.	Max.	Units
Address setup time Address hold time	CD	t_{AS80} t_{AH80}		0 20	--	ns
System cycle time 8 bits bus (read) (write) 4 bits bus (read) (write)	WR0, WR1	t_{CY80}		140 140 140 140	--	
Pulse width 8 bits (read) 4 bits	WR1	t_{PWR80}		65 65	--	
Pulse width 8 bits (write) 4 bits	WR0	t_{PWW80}		35 35	--	
High pulse width 8 bits bus (read) (write) 4 bits bus (read) (write)	WR0, WR1	t_{HPW80}		65 35 65 35	--	
Data setup time Data hold time	D0 to D7	t_{DS80} t_{DH80}		30 20	--	
Read access time Output disable time	D0 to D7	t_{ACC80} t_{OD80}	CL=100pF	-- 12	60 20	
Chip select setup time	CS	t_{CSSA80} t_{CSSD80} t_{CSH80}		10 10 20	--	



Parallel Bus Timing Characteristics (for 8080 MPU)

C/D: 0: Control, 1: Data
W/R: 0: Write Cycle, 1: Read Cycle
Useful Data bits
– Don't Care

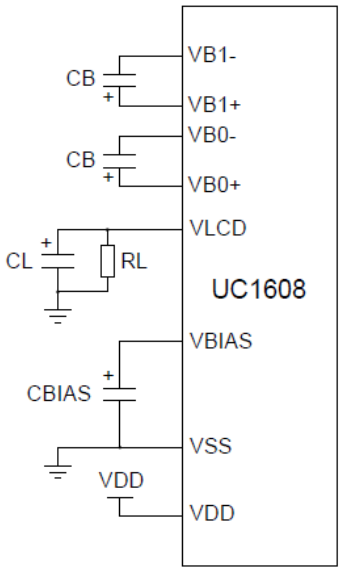
	Command	C/D	W/R	D7	D6	D5	D4	D3	D2	D1	D0	Action	Default
1	Write Data Byte	1	0	#	#	#	#	#	#	#	#	Write 1 byte	N/A
2	Read Data Byte	1	1	#	#	#	#	#	#	#	#	Read 1 byte	N/A
3	Get Status	0	1	BZ	MX	DE	RS	WA	GN1	GN0	1	Get Status	N/A
4	Set Column Address LSB	0	0	0	0	0	0	#	#	#	#	Set CA[3:0]	0
	Set Column Address MSB	0	0	0	0	0	1	#	#	#	#	Set CA[7:4]	0
5	Set Mux Rate and Temperature Compensation	0	0	0	0	1	0	0	#	#	#	Set {MR, C[1:0]}	MR: 1 TC: 00b
6	Set Power Control	0	0	0	0	1	0	1	#	#	#	Set PC[2:0]	101b
7	Set Adv. Program Control (double byte command)	0	0	0	0	1	1	0	0	0	R	For UltraChip only. Do not use.	N/A
		0	0	#	#	#	#	#	#	#	#		
8	Set Start Line	0	0	0	1	#	#	#	#	#	#	Set SL[5:0]	0
9	Set Gain and Potentiometer (double byte command)	0	0	1	0	0	0	0	0	0	1	Set {GN[1:0], PM[5:0]}	GN=3 PM=0
		0	0	#	#	#	#	#	#	#	#		
10	Set RAM Address Control	0	0	1	0	0	0	1	#	#	#	Set AC[2:0]	001b
11	Set All-Pixel-ON	0	0	1	0	1	0	0	1	0	#	Set DC[1]	0=disable
12	Set Inverse Display	0	0	1	0	1	0	0	1	1	#	Set DC[0]	0=disable
13	Set Display Enable	0	0	1	0	1	0	1	1	1	#	Set DC[2]	0=disable
14	Set Fixed Lines	0	0	1	0	0	1	#	#	#	#	Set FL[3:0]	0
15	Set Page Address	0	0	1	0	1	1	#	#	#	#	Set PA[3:0]	0
16	Set LCD Mapping Control	0	0	1	1	0	0	#	#	#	#	Set LC[3:0]	0
17	System Reset	0	0	1	1	1	0	0	0	1	0	System Reset	N/A
18	NOP	0	0	1	1	1	0	0	0	1	1	No operation	N/A
19	Set LCD Bias Ratio	0	0	1	1	1	0	1	0	#	#	Set BR[1:0]	10b=12
20	Reset Cursor Mode	0	0	1	1	1	0	1	1	1	0	AC[3]=0, CA=CR	N/A
21	Set Cursor Mode	0	0	1	1	1	0	1	1	1	1	AC[3]=1, CR=CA	N/A
22	Set Test Control (double byte command)	0	0	1	1	1	0	0	1	TT		For UltraChip only. Do not use.	N/A
		0	0	#	#	#	#	#	#	#	#		

Note: Please refer to UC1608 datasheet for details.

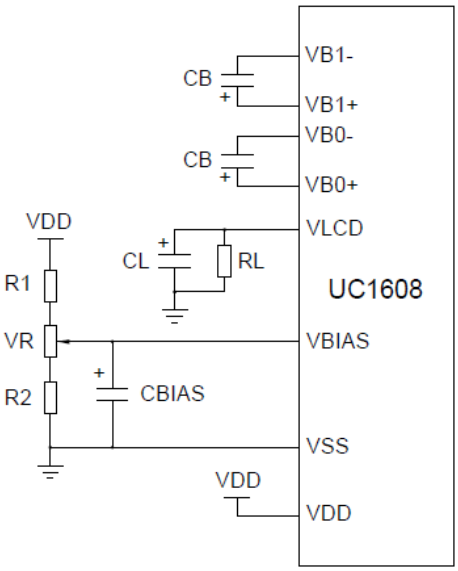
GLCD의 커맨드 리스트.

3.8 Power Supply for Logic and LCD Driving

A. Use internal bias source

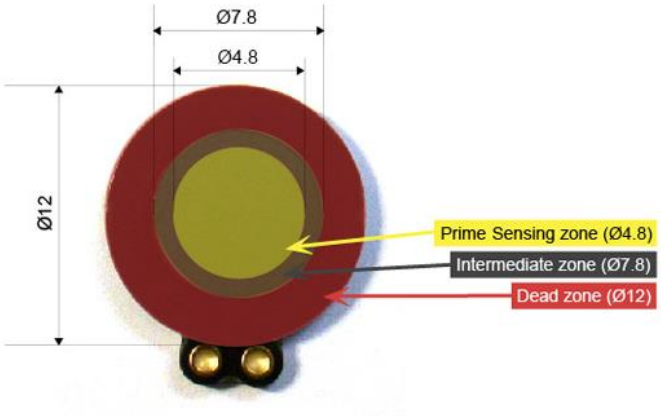


B. Use exernal bias source



1. CB=4.7uF/6.3V, CL=0.1uF/25V, CBIAS=0.1uF/6.3V
2. R1=330KΩ± 1%, R2=91KΩ± 1%, VR=100KΩ± 1%, RL=10MΩ
3. When use internal bias source, LCD contrast can only be adjusted by software.
4. When use external bias source, LCD contrast can be adjusted by either VR or software.
5. To ensure consistency of LCD contrast, circuitry B is recommended prior to circuitry A.

FSR RA12P 압력 센서



Sensing area(㎜)

V_{IN} 에 대한 특별한 언급이 없는 것으로 보아, 5V Vcc를 인가해도 무리가 없을 듯하다.

센서 기판의 V패드와 A패드

단자는 2개가 존재하는데, 한쪽 단자('V'라고 표기)에는 전압이,
다른 한쪽('A'라고 표기'에는 ADC 포트와 연결)

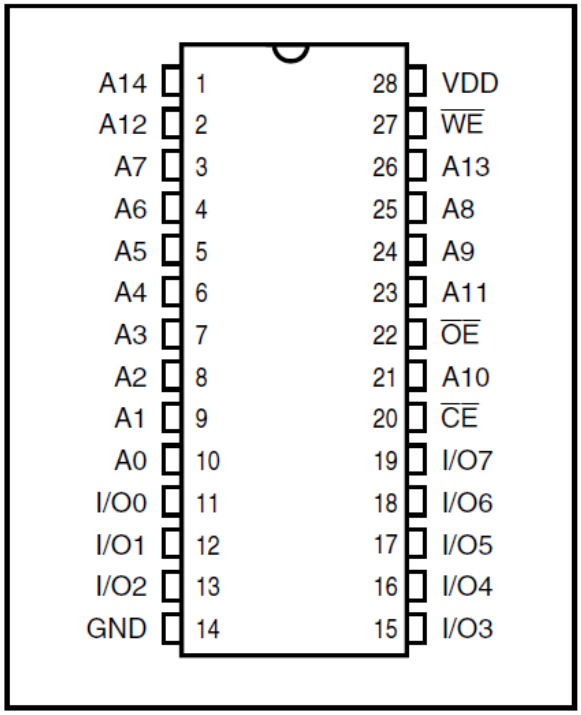
Vin 과 ADC를 연결해줘야 동작하며,
센서 기판에서 Vin은 'V'로 표기되어 있고, ADC는 'A'로 표기

Vin은 아래 세군데의 V패드 중 어느 V패드에 연결해도 됩니다.
세개의 V패드는 내부적으로 연결 되어 있기 때문에

마찬가지로 세군데의 A패드 중에 어느 A 패드에 연결해도 됩니다.

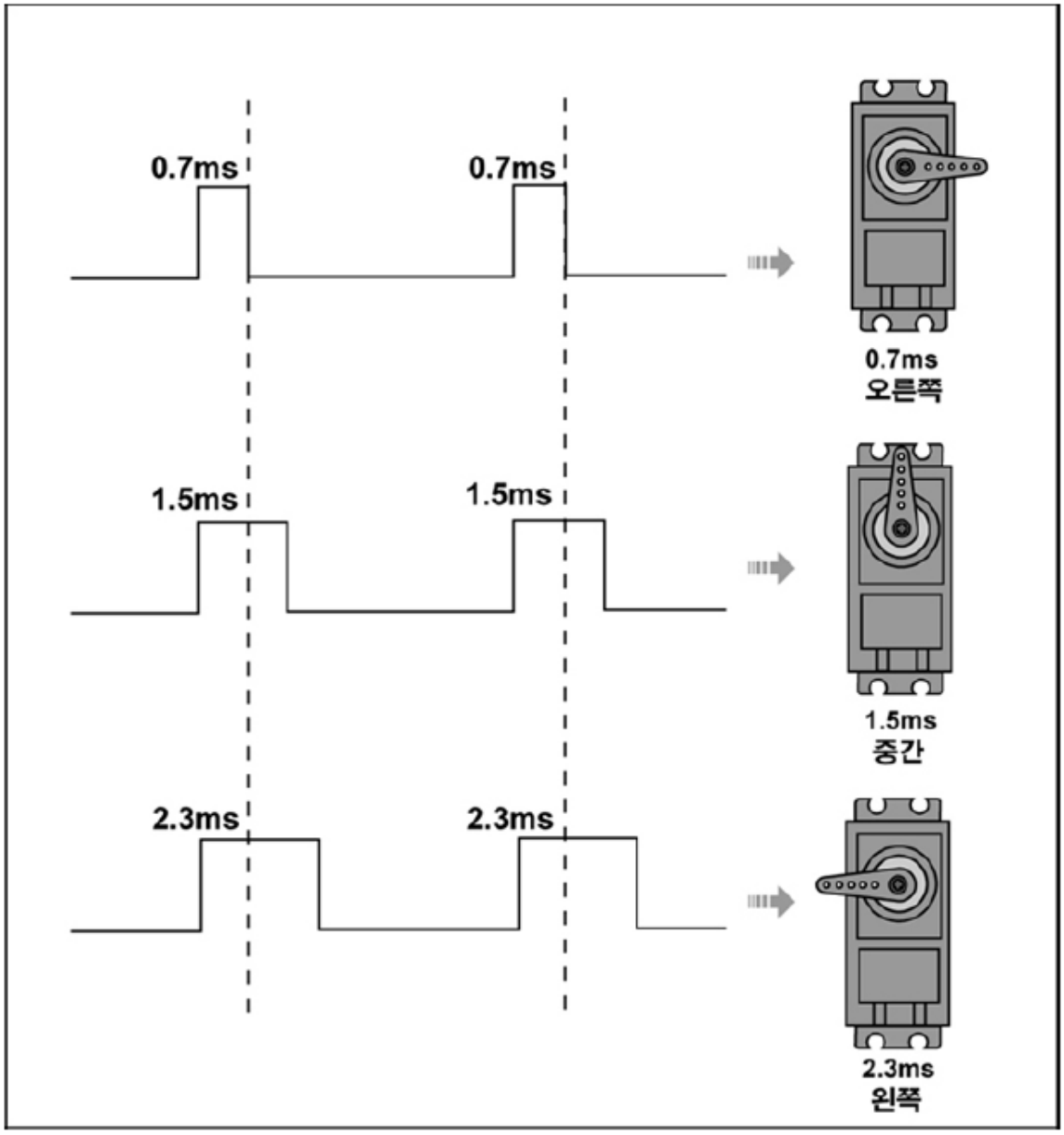
SRAM

PIN CONFIGURATION
28-Pin SOP

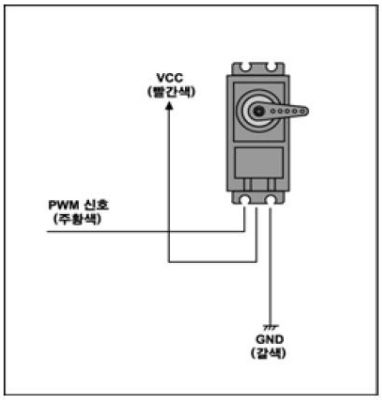


전형적인 SRAM이다. 15비트 주소 입력과 8비트 데이터 입출력 포트를 갖는다. 5V 전원이 공급되어야 하며, 최대 응답 속도는 45ns (22.2MHz) 로 ATMEGA128에 충분히 사용 가능.

서보 모터 HES-288

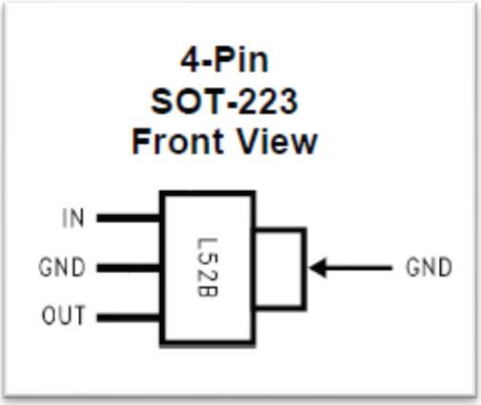


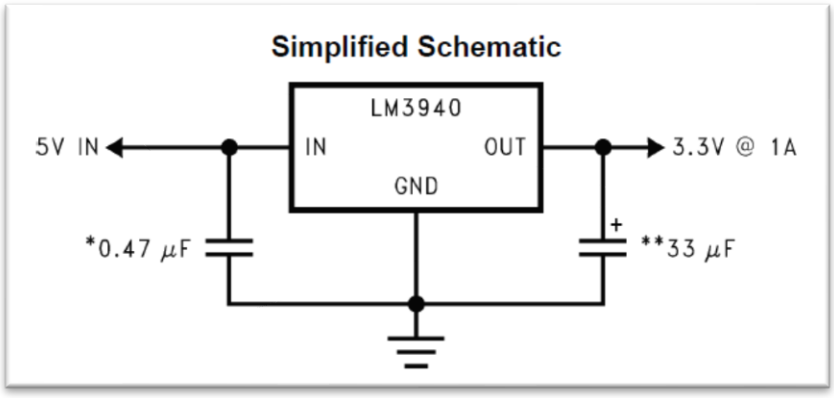
PWM을 이용해 제어되는 서보 모터이다. 게임이 1초에 60번씩 시뮬레이션 되므로, 16.67ms마다 모터에 대해 갱신 신호가 들어간다. OCR1B 비교 매치 인터럽트에서 OC1B 출력을 클리어하는 방법으로, PWM 출력을 소프트웨어 생성



정전압 레귤레이터 LM3940 SOT-223

GLCD 디바이스의 바이어스 전압인 3.3V를 만들어내기 위해 사용되는 정전압 레귤레이터이다. 위와 같은 핀 배치를 가지며, 회로상에는 다음과 같이 바이어스되어야 한다.

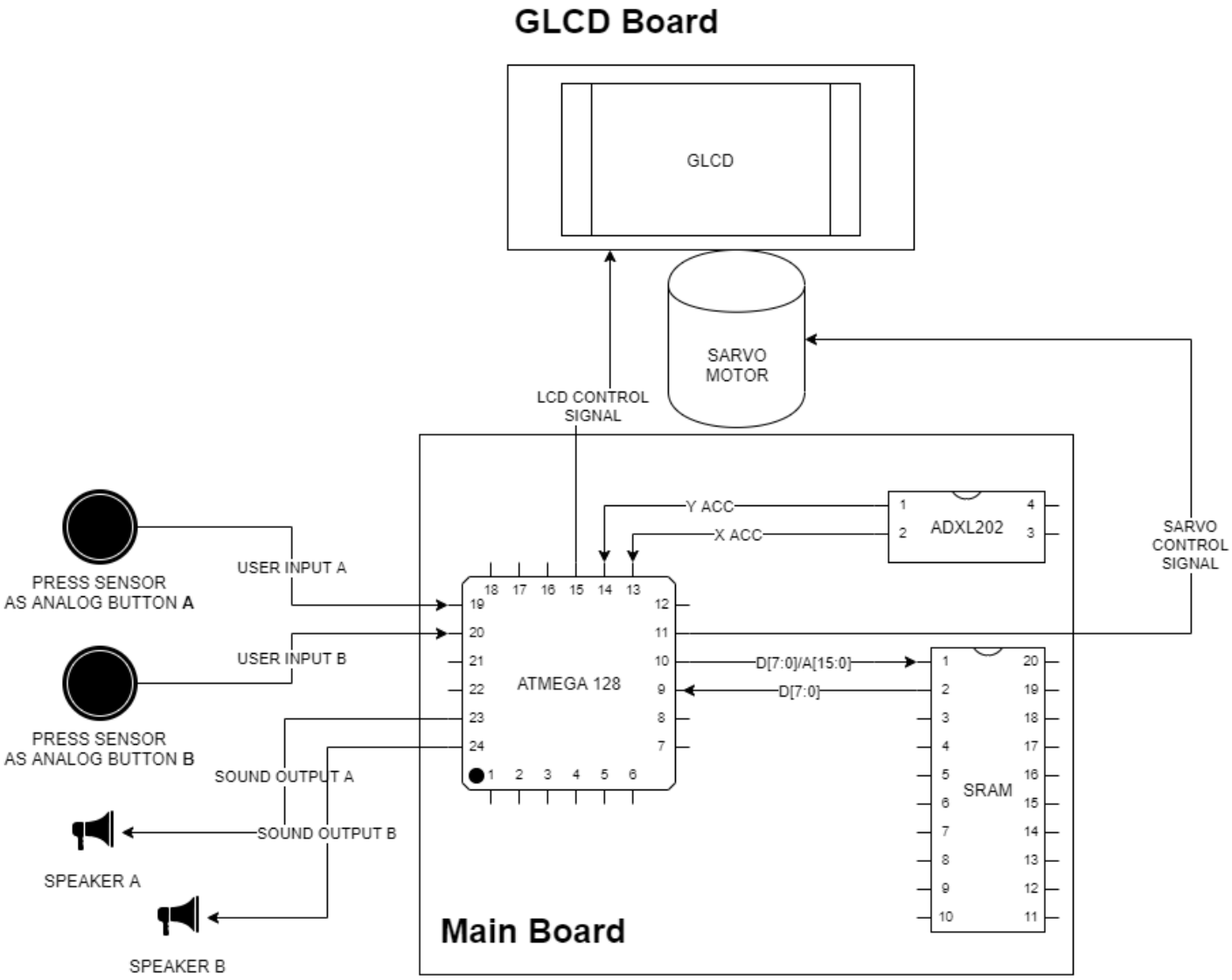




1A의 전류를 출력할 수 있는데, 이 디바이스에 연결되는 LCD의 V_{DD} 소모 전류량은 최대 1.5mA에 불과하므로 크게 문제되진 않는다.

4. 작품 구성도 및 회로도

작품 구성도



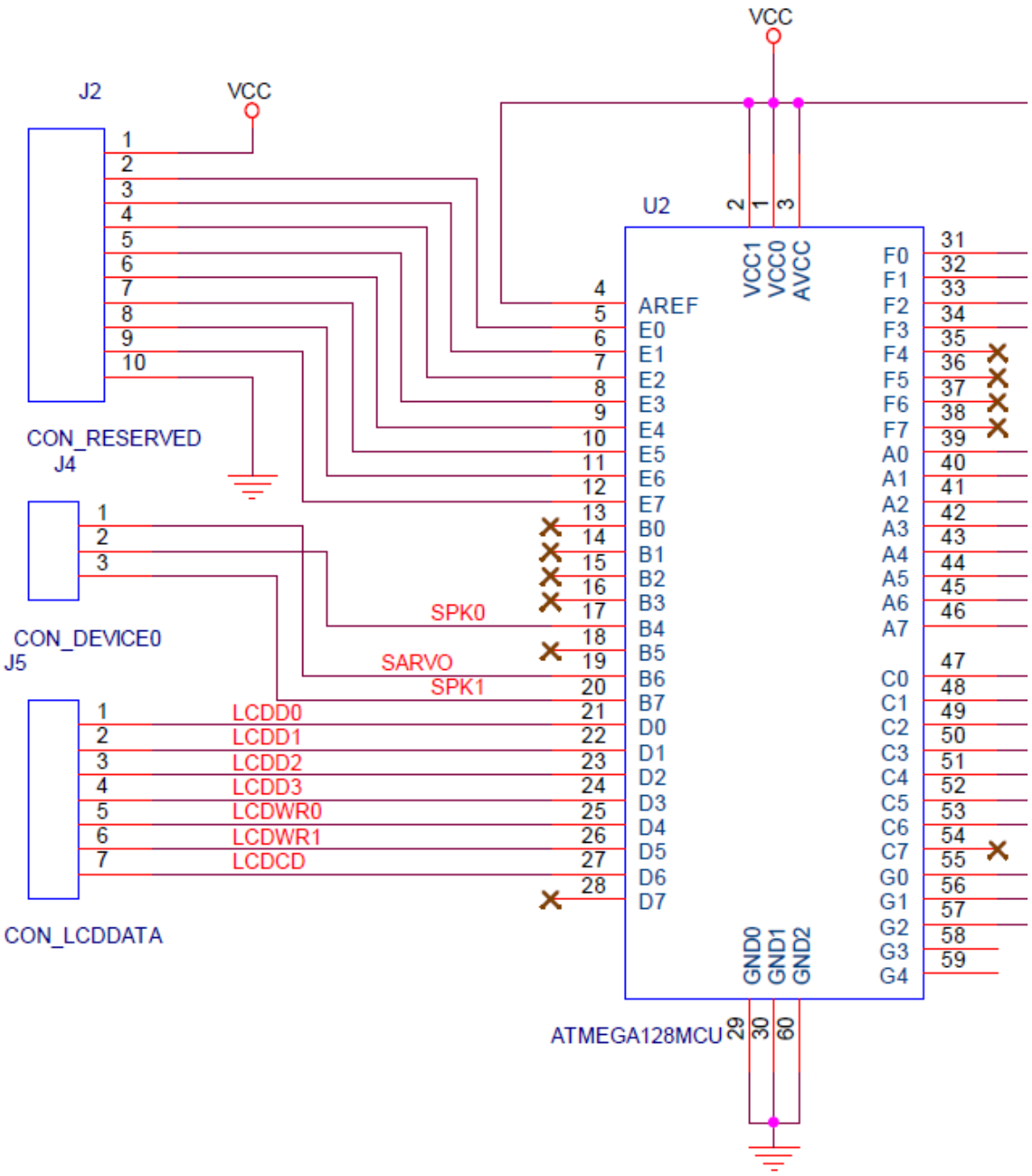


Figure 1. 외부 커넥터 연결

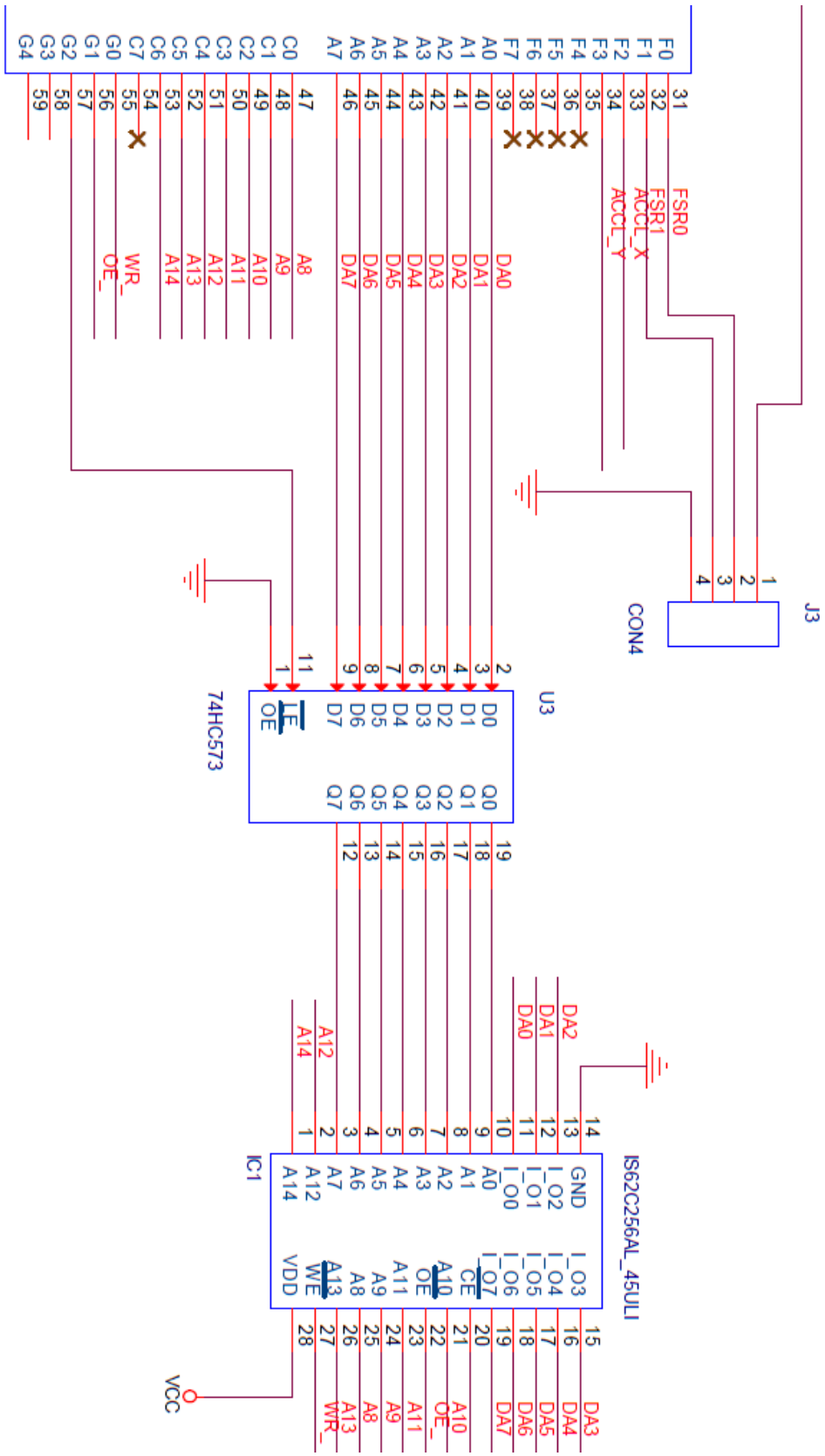


Figure 2. 메모리 연결

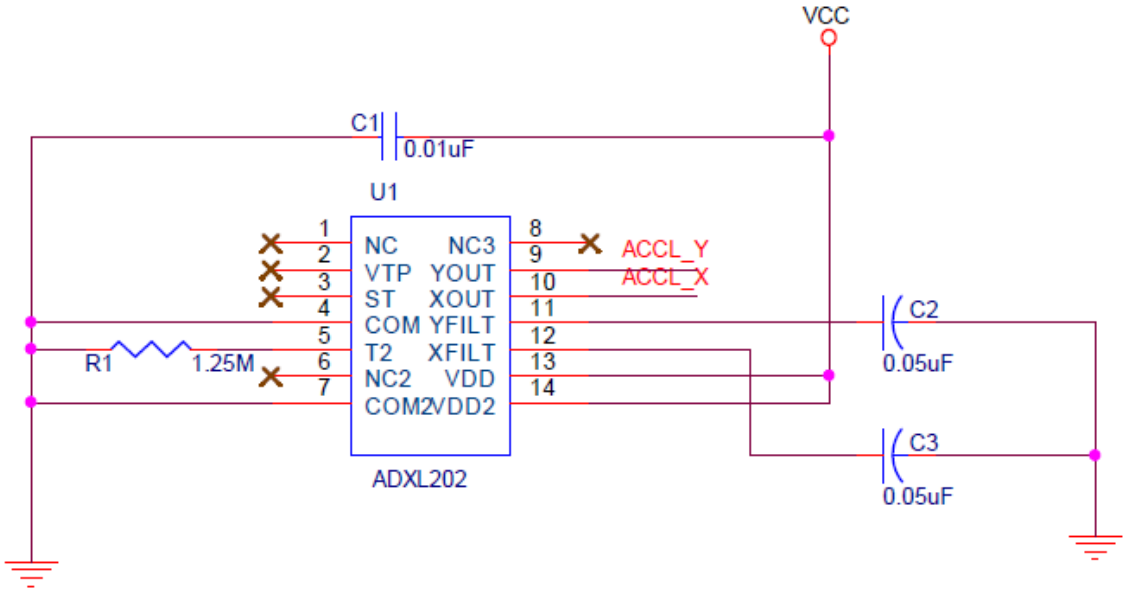


Figure 3. 가속도 센서

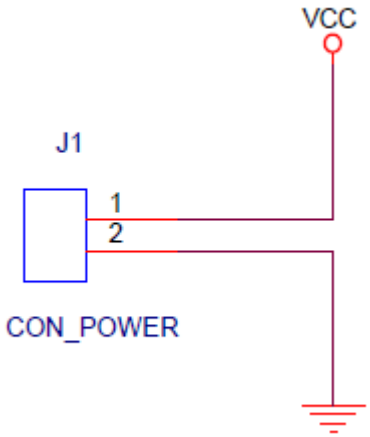


Figure 4. 전원 커넥터

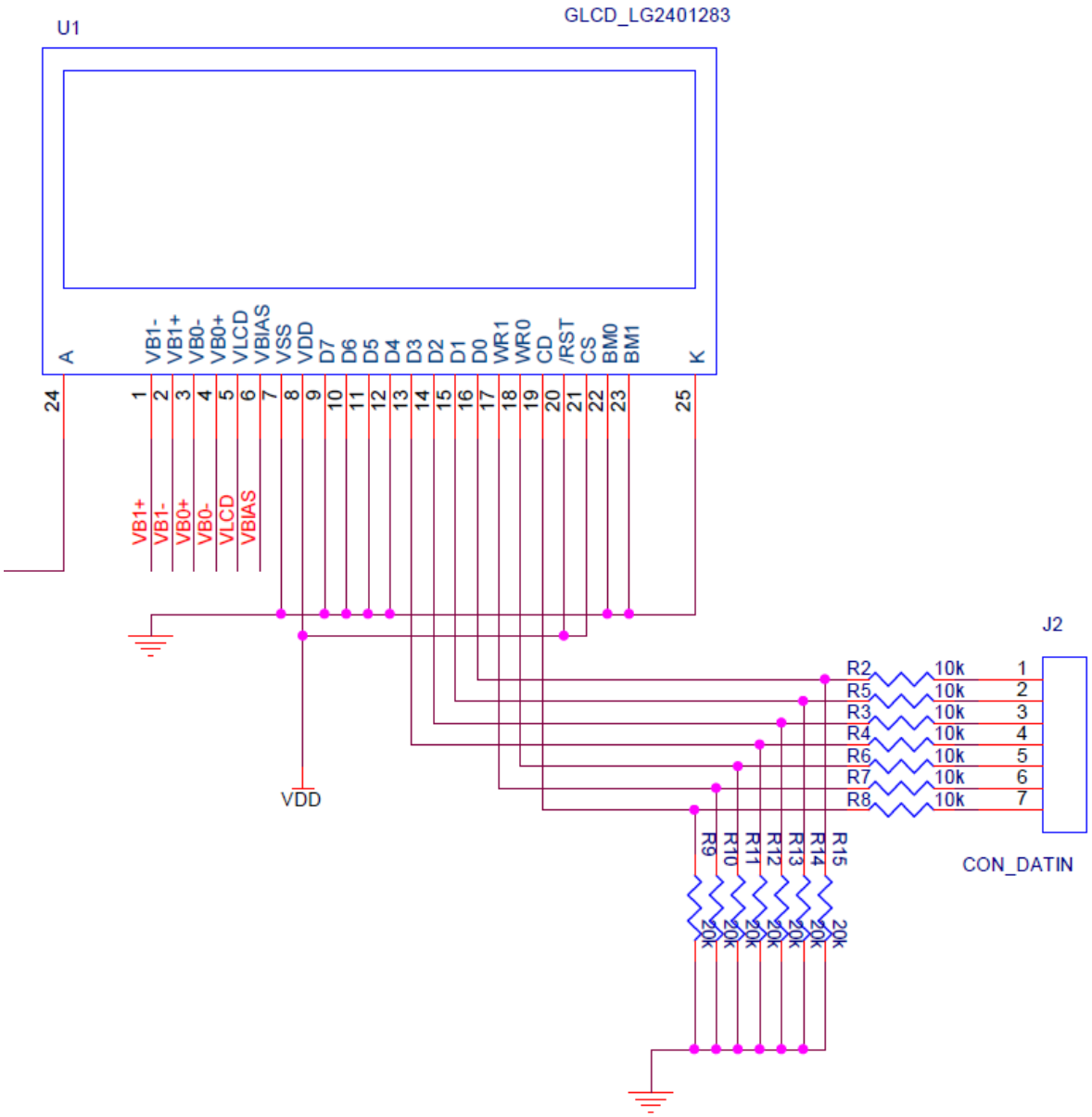


Figure 5. 커넥터 연결.

GLCD은 신호 입력 전압 또한 3.3V로 맞추어야 하므로, 5V의 출력 전압을 위와 같이 저항을 이용해 분배, 3.3V로 바이어스하였다.

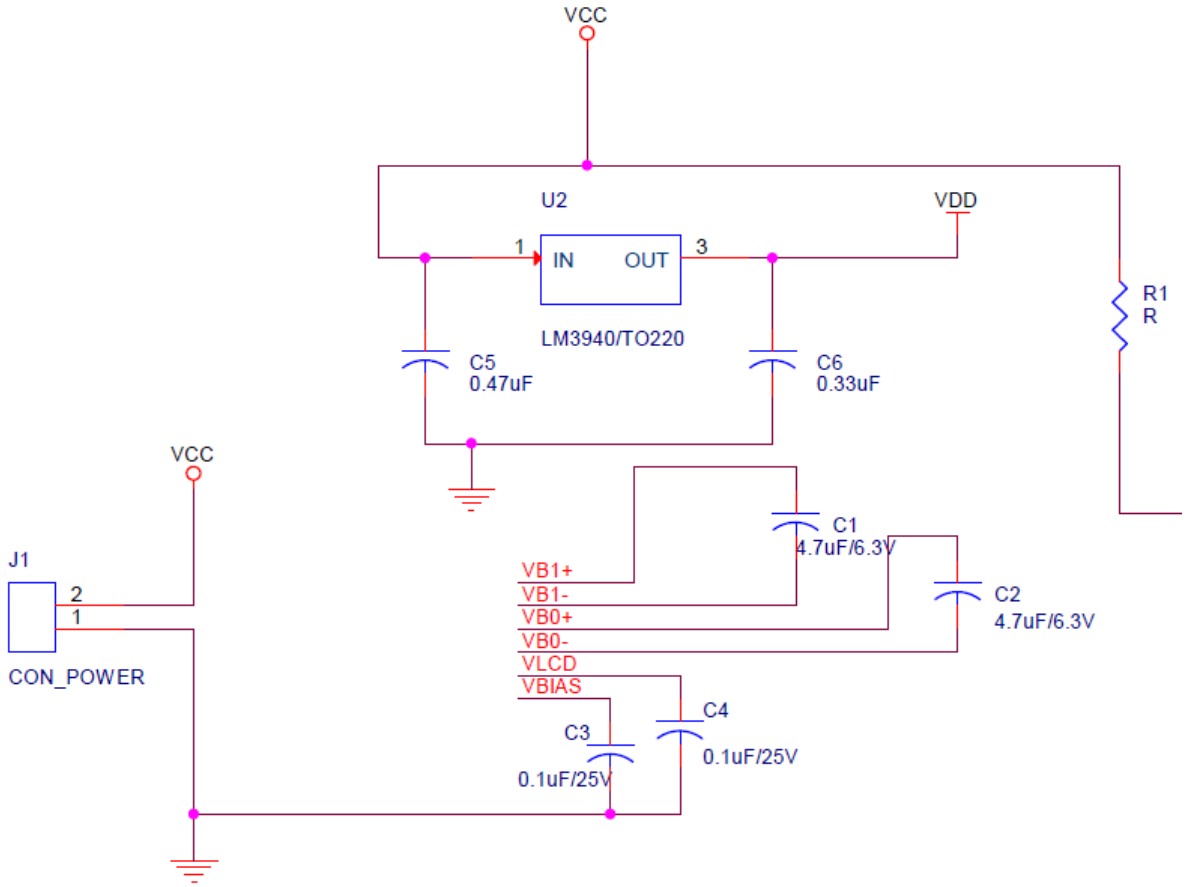


Figure 6. 바이어스

5. 작품 진행상황

PCB기판이 인쇄되었고, 모든 부품을 연결한 뒤 정상적으로 동작하는 것을 확인하였습니다.

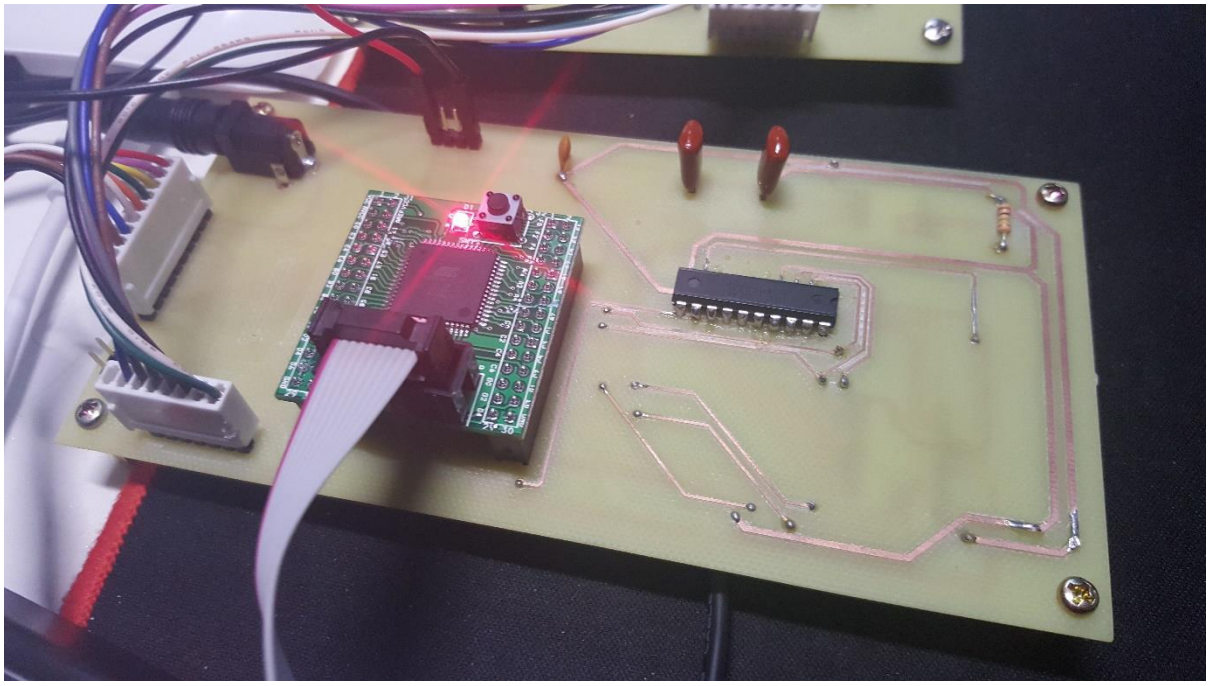


Figure 7. 메인 보드

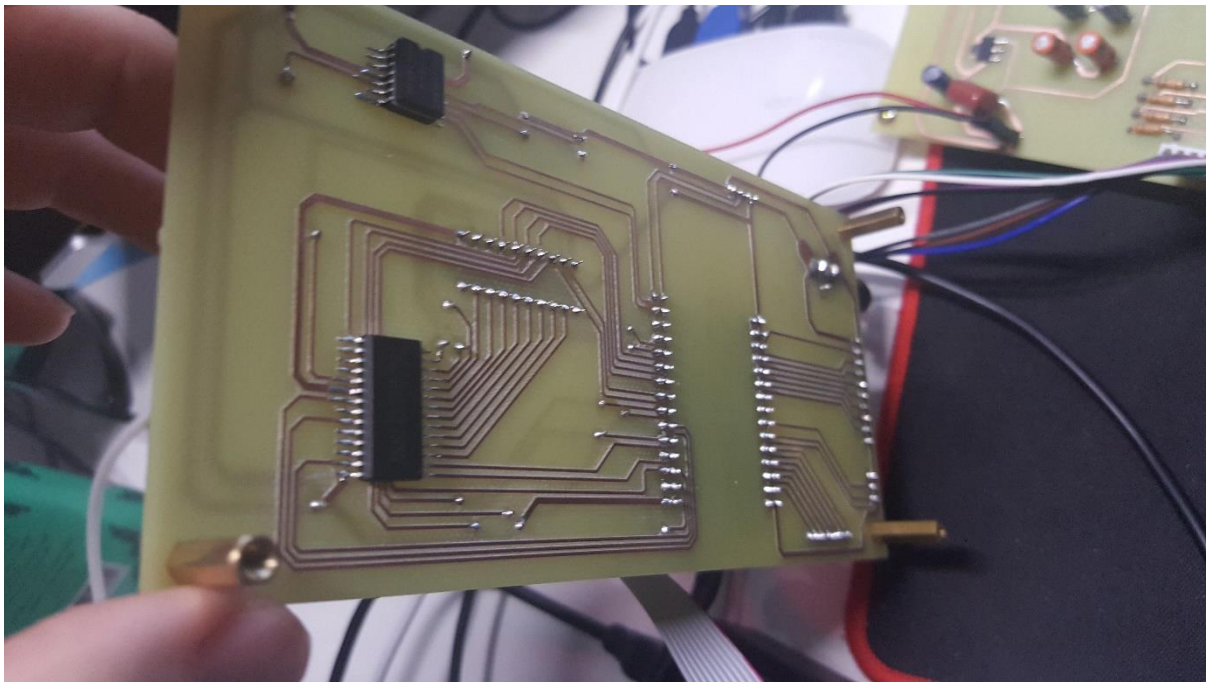


Figure 8. 메인보드 후면

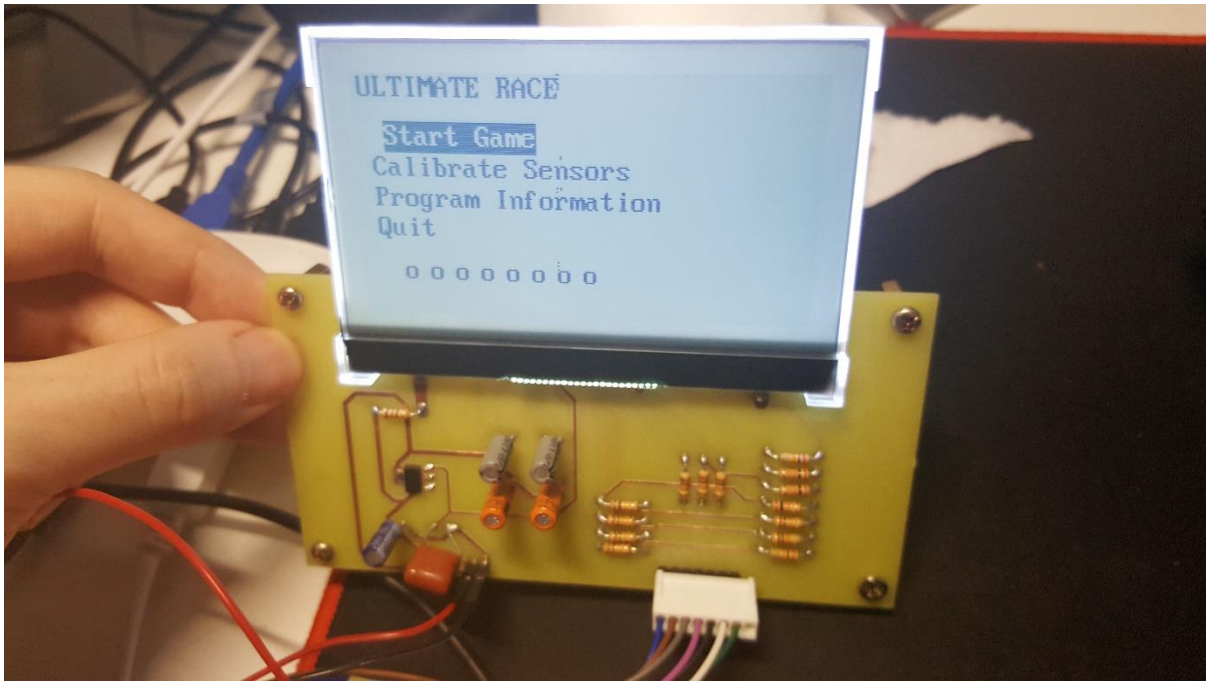


Figure 9. 디스플레이

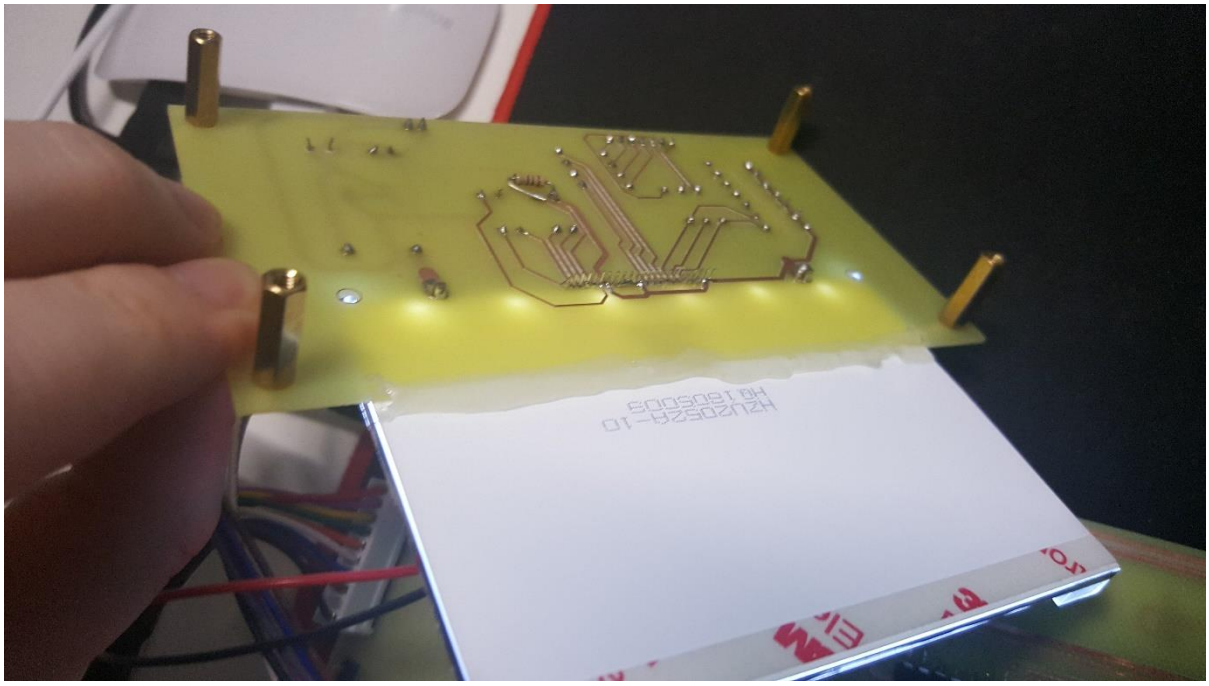


Figure 10. 디스플레이 후면

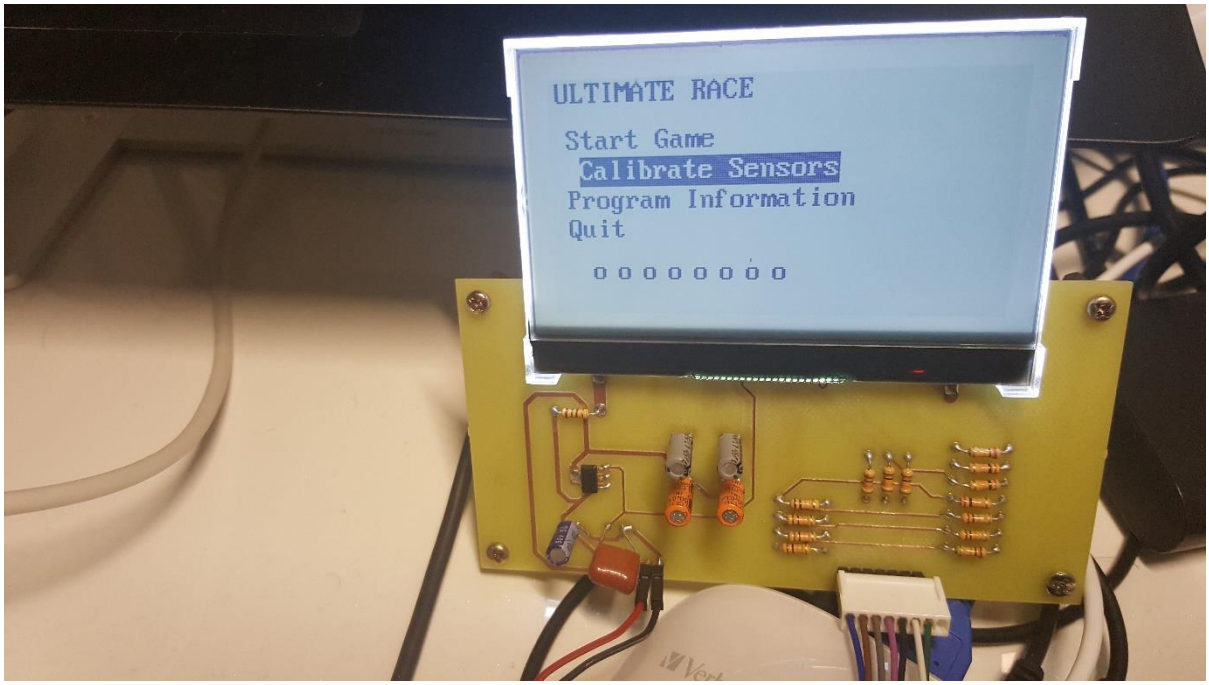


Figure 11. 메뉴 선택

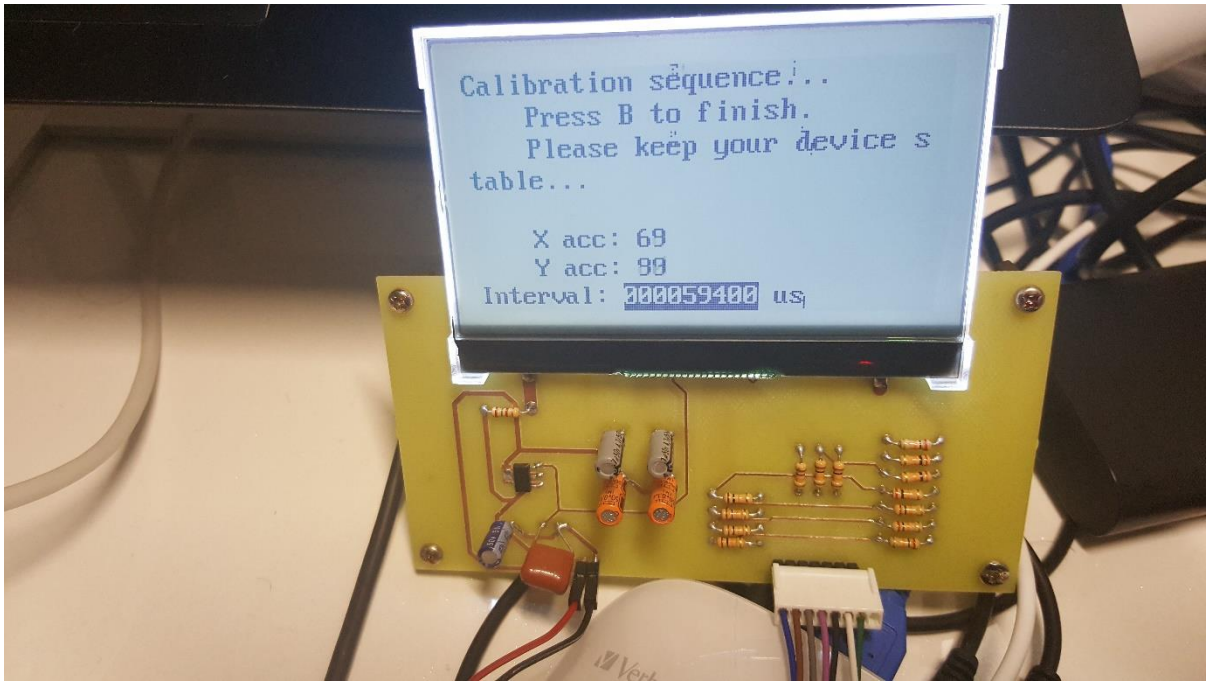


Figure 12. 가속도 센서 조정

```
void main( void )
{
    void InitializeDevice();
    void runTest();

    InitializeAnalogDevice();
    InitializeDevice();
    // runTest();

    // PROGRAM INITIALIZATION
    gSession.Update = nullfunc;
    gSession.Draw = nulldraw;
    gSession.data__ = NULL;

    INITSESSION_MAIN();

    byte RenderingInterval = 0;
    // MAIN PROGRAM LOOP
    while ( 1 )
    {
        UpdateTimer();
        UpdateInputStatus();

        gSession.Update();
        gSession.Draw( RenderingInterval == 0 );

        RenderingInterval =
            RenderingInterval == 0
            ? TARGET_RENDER_FRAME_INTERVAL - 1
            : RenderingInterval - 1;

        LCDDevice__Render();

        while ( !GOOD_TO_UPDATE );
        GOOD_TO_UPDATE = false;
    }
}
```

프로그램 루프입니다. 매 500us마다 활성화되는 타이머 인터럽트가 총 66번, 즉 33ms가 될 때마다 GOOD_TO_UPDATE 플래그를 활성화시키고, 이것으로 프로그램 루프의 실행을 1초에 30번으로 제한합니다. 아래는 프로그램 주기를 제어하는 타이머 인터럽트 함수입니다.

```
 #define TCNT1_SETUP TCNT1 = 0xffff - 7999
ISR( TIMER1_OVF_vect )
{
    TCNT1_SETUP;
    enum { ITER_COUNT = 66 };
    static byte IterCnt = 0;

    ++IterCnt;

    if ( IterCnt == ITER_COUNT )
    {
        GOOD_TO_UPDATE = true;
        IterCnt = 0;
    }

    gButton_Captured |= INPUT_VECTOR;
    UpdateAccel();
}
```

인터럽트 함수의 말미에서 호출되는 UpdateAccel 함수는 펄스 폭을 통해 기울기를 나타내는 가속도 센서의 출력을 측정하기 위해 매 500us마다 호출되는 함수입니다. 매 Tick마다 x축과 y축의 기울기를 측정하며, 출력 신호의 Rising edge를 감지하게 되면 한 펄스 주기 전체의 Tick에서 High인 동안 측정된 Tick 개수의 비율을 측정 값으로써 전역 변수에 캐시합니다.

```
byte ACC_PERCENTX;
byte ACC_PERCENTY;
void UpdateAccel()
{
    // X
    static byte xtot = 0;
    static bool xprv = 0;
    static byte xidx = 0;

    bool x = ACC_X;
    if ( x ) {
        if ( x ^ xprv ) {
            // ACC_PERCENTX = ACC_XCNT * 100 / xtot;
            ACC_XARR[xidx++& SAMPLE_MOD] = ACC_XCNT * 100 / xtot;
            uint16 tot = 0;
            byte i = SAMPLE_POW2;
            while ( i-- ) {
                tot += ACC_XARR[i];
            }
            ACC_PERCENTX = tot >> SAMPLE_BITS;
            ACC_XCNT = 1;
            xtot = 0;
        }
        else {
            ++ACC_XCNT;
        }
    }
    ++xtot;
    xprv = x;
```

Y축에 대한 측정은 코드 구조가 완전히 같으므로 생략하였습니다. 특기할 점은, 가속도를 측정할 때 가장 최근의 16개의 측정값을 모두 합한 뒤 그 평균을 반환한다는 점입니다. 가속도 센서의 펄스 주기가 10 ms정도이므로 장치를 기울인 이후 160ms정도가 지나야만 완전한 값을 얻게 되지만, 이를 통해 값이 튀지 않고 자연스럽게 보간되는 장점이 있습니다.

32킬로바이트의 넉넉한 외장 SRAM을 설치했기 때문에, 힙 메모리 영역은 외장 메모리 영역을 사용합니다.

```
void init_ebi_heap( void ) {
    // the malloc heap start and end pointers
    extern char *__malloc_heap_start;
    extern char *__malloc_heap_end;

    // your code to init the ebi goes here

    // set heap start and end
    __malloc_heap_start = (char *) 0x8000;
    __malloc_heap_end = (char *) 0xffff;

    MCUCR |= mask( SRE );
  }
```

```
#define LCD_CD 6
#define LCD_WR1 5
#define LCD_WR0 4
#define LCD_WR LCD_WR0
#define LCD_RD LCD_WR1
#define LCD_D3 3
#define LCD_D2 2
#define LCD_D1 1
#define LCD_D0 0
#define LCD_DEFAULT mask(LCD_WR, LCD_RD)

#define LO(DAT) ( DAT & 0x0f )
#define HI(DAT) ( ( DAT & 0xf0 ) >> 4 )

#define LCDCOM_COLUMN_LO(DAT) LO(DAT)
#define LCDCOM_COLUMN_HI(DAT) (HI(DAT)|0X10)
#define LCDCOM_SYSRST 0B11100010
#define LCDCOM_MUXR_TEMPCOMP(MR, C) (0x20|((MR!=0)<<2)|((C)&0B11))
#define LCDCOM_POWERCON(CON) (0B00101000|((CON)&0B111))
#define LCDCOM_STARTLINE(VAL) (0X40|((VAL)&0B111111))
#define LCDCOM_ADDRCTRL(VAL) (0X88|(VAL&0B111))
#define LCDCOM_ALLPXLON(EN) (0XA4|((EN)!=0))
#define LCDCOM_DISPEN(EN) (0XAE|((EN)!=0))
#define LCDCOM_FXLINE(VAL) (0X90|((VAL)&0X0F))
#define LCDCOM_PGADDR(VAL) (0XB0|((VAL)&0X0F))
#define LCDCOM_MAPCTRL(VAL) (0XC0|((VAL)&0X0F))
#define LCDCOM_BIASRATIO(VAL) (0XC8|((VAL)&0B11))
#define LCDCOM_GAIN_POTENTIAL_INIT 0X81
#define LCDCOM_GAIN_POTENTIAL_VAL(VAL) (VAL)
#define LCDCOM_DISPINVERT(EN) (0XA6|((EN)!=0))

#define LCDOUTPUT(DAT) PORTD = (DAT); _delay_us(1)

static void COMMAND( uint8 data ) {
    // Refresh 4-bit latch
    uint8 PutDat;
    PutDat = ( LCD_DEFAULT | HI( data ) ) & ~mask( LCD_WR );
    LCDOUTPUT( PutDat );
    PutDat |= mask( LCD_WR );
    LCDOUTPUT( PutDat );

    PutDat = ( LCD_DEFAULT | LO( data ) ) & ~mask( LCD_WR );
    LCDOUTPUT( PutDat );
    PutDat |= mask( LCD_WR );
    LCDOUTPUT( PutDat );
}
static void DATAWR( uint8 data ) {
    // Refresh 4-bit latch
    uint8 PutDat;
    PutDat = ( mask( LCD_CD ) | LCD_DEFAULT | HI( data ) ) & ~mask( LCD_WR );
    LCDOUTPUT( PutDat );
    PutDat |= mask( LCD_WR );
    LCDOUTPUT( PutDat );

    PutDat = ( mask( LCD_CD ) | LCD_DEFAULT | LO( data ) ) & ~mask( LCD_WR );
    LCDOUTPUT( PutDat );
    PutDat |= mask( LCD_WR );
    LCDOUTPUT( PutDat );
}
```

GLCD를 구동하기 위한 LCD 드라이버를 만들었습니다. 4개의 데이터 핀만 사용하기 때문에 모든 명령어와 데이터는 두 번에 걸쳐 송신하게 됩니다. 4바이트의 선후 순서는 매번 커맨드와 데이터 비트가 바뀔 때마다 리셋되므로, 프로그램 초기화 단계에서 이 순서를 갱신하는 명령어 또한 정의하였습니다.

```
static void REFRESH()
{
    LCDOUTPUT( LCD_DEFAULT | mask( LCD_CD ) );
    LCDOUTPUT( LCD_DEFAULT );
}
```

LCD의 초기화는 데이터시트에 나온 기본 초기화 순서를 따르며, 일부 GLCD의 출력 방향을 맞추기 위한 몇 가지 프로퍼티만을 임의로 변경하였습니다.

```
#define INITIALIZATION_DELAY_MS 4
#define COMMAND_DELAY(VAL) COMMAND( VAL ); _delay_ms( INITIALIZATION_DELAY_MS );
    // LCD Initialization
    _delay_ms( 1000 );
    COMMAND_DELAY( LCDCOM_SYSRST );
    _delay_ms( 1000 );
    COMMAND_DELAY( 0x26 );
    COMMAND_DELAY( 0x2d );
    COMMAND_DELAY( 0xea );
    COMMAND_DELAY( 0x81 );
    COMMAND_DELAY( 0x8b );
    COMMAND_DELAY( LCDCOM_MAPCTRL( 0b1001 ) );
    COMMAND_DELAY( 0x40 );
    COMMAND_DELAY( LCDCOM_ADDRCTRL( 0b001 ) );
    COMMAND_DELAY( LCDCOM_DISPEN( true ) );
```

위에 해당하는 명령어들을 순서대로 송출하여 GLCD의 초기화를 하게 됩니다.

```
void LCDDevice__Render()
{
    byte i, j;
    byte const* lpBuff;
    for ( i = 0; i < LCD_NUM_PAGE; ++i )
    {
        lpBuff = LCDBuffer + i; // X PIVOT
        for ( j = 0; j < LCD_NUM_COLUMN; ++j, lpBuff += LCD_NUM_PAGE )
        {
            byte dat = *lpBuff;
            DATAWR( dat );
        }
    }
}
```

실제 GLCD에 데이터를 출력할 때 사용되는 함수입니다. LCD의 각 행이 페이지를, 각 열이 컬럼을 나타내는데, 실제 데이터를 출력하게 되면 커서가 컬럼 방향을 향해 증가하게 되므로, 바깥쪽 루프에서 페이지 개수만큼 오프셋하고, 거기에서 포인터를 매 데이터 송출마다 PAGE 개수(여기에선, 128/8 == 16개입니다)만큼 증가시키게 됩니다.

이처럼 모든 화면 데이터가 VBuffer에 기록되어야 뒤 화면에 출력되므로, VBuffer를 조작하는 다수의 함수성이 정의되어 있습니다.

```
inline void VBuffer_DrawDot( int16 y, int16 x )
void VBuffer_DrawChar( byte xCol, byte y, char ASCII_IDX, bool bInversed );
void VBuffer_DrawString( byte* xCol, byte* y, const char* String, bool bInversed );
void VBuffer_DrawLine( int16 xbeg, int16 ybeg, const int16 xend, const int16 yend );
```

DrawLine 함수는 브레젠험의 직선 그리기 알고리즘을 사용합니다. DrawChar는 CGROM이라 명명된, 글자 하나마다 8*16 비트가 할당된 데이터 공간에서 글자 그래픽 데이터를 읽어와 VBuffer에 복사합니다.

DrawString은 DrawChar를 문자열의 개수만큼 호출하는데, 포인터의 형태로 전달된 x와 y 커서 오프셋을 증감시켜 다음 문자가 위치해야 하는 자리로 이동시킵니다. 특히, CR이나 LF, 탭 등의 특수 문자 몇 가지를 지원합니다.

```
typedef void( *FSessionEventSignature )( );
typedef void( *FSessionDrawEventSignature )( bool );

typedef struct {
    FSessionEventSignature Update;
    FSessionDrawEventSignature Draw;
    void* data__; // Should be dynamically allocated.
} FSessionState;
```

프로그램의 상태는 '세션'으로 정의되며, 업데이트 콜백과 드로우 콜백, 그리고 각 세션의 고유한 정보를 담을 수 있는(필수는 아님) 무형식의 포인터로 구성되어 있습니다. 업데이트 콜백은 입력 처리와 게임 로직 업데이트를 담당하며, 드로우 콜백은 화면 출력을 위한 VBuffer의 갱신을 맡습니다. 단, 차후 무거운 렌더링 작업에 있어 프레임 사이에 인터벌을 두는 방식으로 간격을 조절할 수 있도록 하기 위해, 파라미터로 몇 프레임마다 한 번씩만 true가 전달되게끔 해 두었습니다. 이 파라미터는 optional합니다.

```
extern FSessionState gSession;
```

각 세션 사이의 전이는 이 전역 프로그램 세션의 내부 멤버들을 교환하는 것을 통해 이루어지며, 이 과정은 주로 세션 전이 절차를 캡슐화한 아래와 같은 함수에 의해 일괄적으로 수행됩니다. 세션 전이는 몇 가지 멤버를 교체하는 것만으로 이루어지므로 그 자체에 큰 오버헤드는 없습니다.

```
void INITSESSION_MAIN()
{
    FMainScreenInfo* lpSessionInfo = memset(
        Malloc( sizeof( FMainScreenInfo ) ),
        0,
        sizeof( FMainScreenInfo )
    );

    SetSessionData( lpSessionInfo );

    gSession.Draw = main_draw;
    gSession.Update = main_update;
}
```

현재까지는 예의 사진과 같이 메인 화면과 센서 조정 화면에 대해서만 세션이 구현되어 있습니다.

```
static void main_update(){
    byte* cursor = &( (FMainScreenInfo*) gSession.data__ )->Cursor;
    if ( gButton_Pressed & mask( BUTTON_D, BUTTON_R ) ) {
        *cursor = ++( *cursor ) == ARRAYCOUNT( MainMenuStrings ) ? 0 : *cursor;
    }
    if ( gButton_Pressed & mask( BUTTON_U, BUTTON_L ) ) {
        *cursor = ( *cursor )-- == 0 ? ARRAYCOUNT( MainMenuStrings ) - 1 : *cursor;
    }
    if ( gButton_Pressed & mask( BUTTON_A ) ) {
        switch ( *cursor ) {
        case 0: break;
        case 1: {
                gSession.Draw = main_calib_draw;
                gSession.Update = main_calib_update;
                break;
            }
        case 2: break;
        case 3: {
                LCDDevice__HardReset();
                _delay_ms( 1000 );
                *cursor = 0;
                while ( 1 );
                LCDDevice__Initialize();
            }
        default:
            break;
        }
    }
}
```

이처럼, 업데이트 함수는 입력을 처리하고, 처리된 입력에 대하여 각종 로직의 처리를 합니다.

```
static void main_draw( bool complxDraw ) {
    VBuffer_Clear();

    byte cursor = ( (FMainScreenInfo*) gSession.data__ )->Cursor;
    byte x = 0, y = 4;
    int8 i;

    VBuffer_DrawString( &x, &y, "ULTIMATE RACE\r\n", false );
    x += 1;
    for ( i = 0; i < ARRAYCOUNT( MainMenuStrings ); ++i, x += 2 ) {
        y = i == cursor ? 16 : 9;
        VBuffer_DrawString( &x, &y, MainMenuStrings[i], i == cursor );
    }

    byte inp = gButton_Hold;
    x += 1;
    y = 8;
    for ( i = 0; i < 8; ++i ) {
        VBuffer_DrawChar( x, y += 14, 'o', inp & 1 );
        inp >>= 1;
    }
  }
```

이는 메인 화면의 그리기 콜백으로, 커서 위치에 따라 선택된 엘리먼트를 강조 처리합니다.

디지털 버튼의 입력은 폴링으로 처리되며, 프레임 사이 33ms의 인터벌에서 혹시라도 놓치는 일이 생기지 않도록, 매 500us마다 호출되는 타이머 인터럽트 내에서 먼저 모든 버튼의 입력이 캡쳐됩니다.

```
#define INPUT_VECTOR (~PINE & 0x7f)
```

현재는 PINE로 입력이 들어옵니다. 이후 입력 배열이 변경되어도 손쉽게 변경할 수 있게끔 매크로 처리하였습니다.

```
gButton_Captured |= INPUT_VECTOR;
```

타이머 인터럽트에서 위와 같이 모든 입력을 캡쳐합니다.

```
void UpdateInputStatus()
{
    static byte Previous;
    byte Input = gButton_Captured;
    byte Delta = Input ^ Previous;
    gButton_Pressed = Delta & Input;
    gButton_Released = Delta & ( ~Input );
    gButton_Hold = Input;
    Previous = Input;
    gButton_Captured = 0;
}
```

현재 프레임에서의 상승 에지와 하강 에지, 그리고 누르고 있는 버튼을 모두 캐시한 후, 캡쳐된 버튼을 초기화합니다. 이 과정은 매 프레임, 메인 프로그램 루프에서 반복됩니다.