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

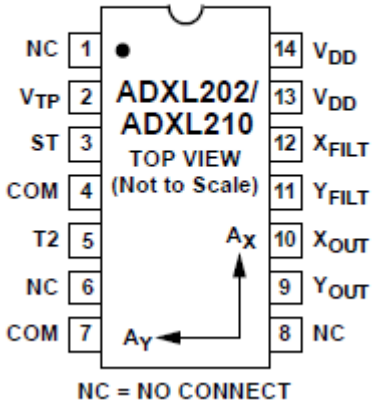
PF2		입력	ADXL 202 X_{OUT}	가속도 X축 PWM 입력
PF3		입력	ADXL 202 Y_{OUT}	가속도 Y축 PWM 입력
PB4	OC0	출력	SPK 0	0번 스피커 주파수 변조 출력
PB7	OC2	출력	SPK 1	1번 스피커 주파수 변조 출력
PB6		출력	HES-288	서보모터 PWM 제어신호
PE0		입출력	출력 커넥터	예약
PE1		입출력	출력 커넥터	예약
PE2		입출력	출력 커넥터	예약
PE3		입출력	출력 커넥터	예약
PE4		입출력	출력 커넥터	예약
PE5		입출력	출력 커넥터	예약
PE6		입출력	출력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

<사용 부품 사양서>

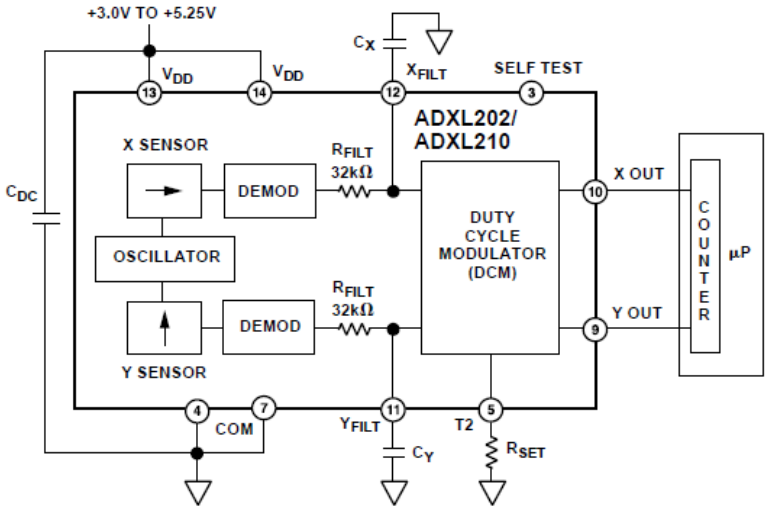
ADXL202 – 디지털 출력 가속도 감지 센서

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.



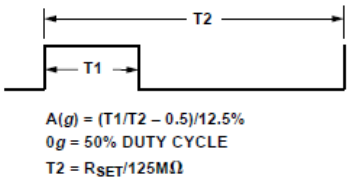


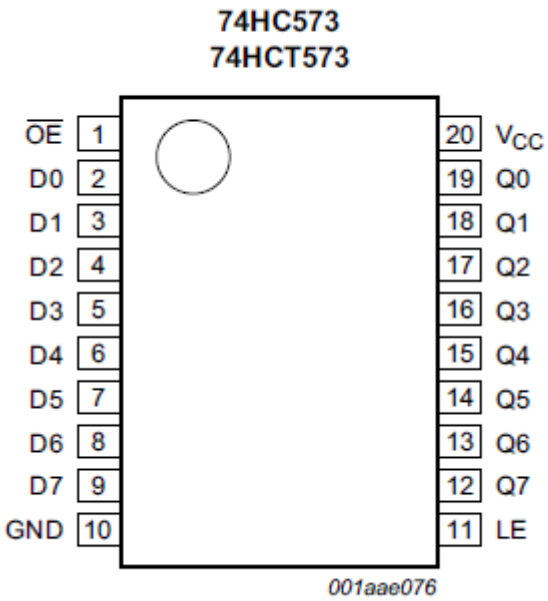
Figure 13. Block Diagram

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.

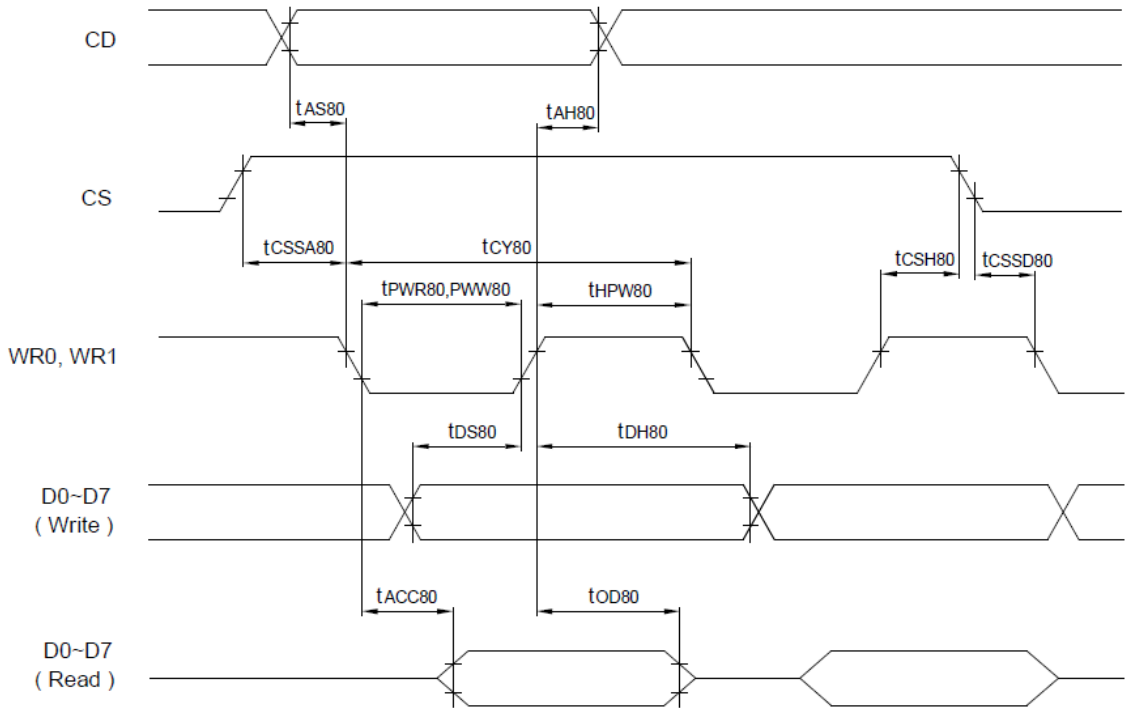
1.3 Terminal Functions

<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

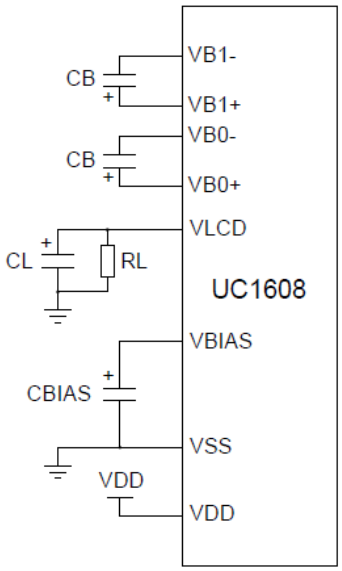
	Command	C/D	W/R	D7	D6	D5	D4	D3	D2	D1	D0	Action	Default
1	Write Data Byte	1	0	#	#	#	#	#	#	#	#	Write 1 byte	N/A
2	Read Data Byte	1	1	#	#	#	#	#	#	#	#	Read 1 byte	N/A
3	Get Status	0	1	BZ	MX	DE	RS	WA	GN1	GN0	1	Get Status	N/A
4	Set Column Address LSB	0	0	0	0	0	0	#	#	#	#	Set CA[3:0]	0
	Set Column Address MSB	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.

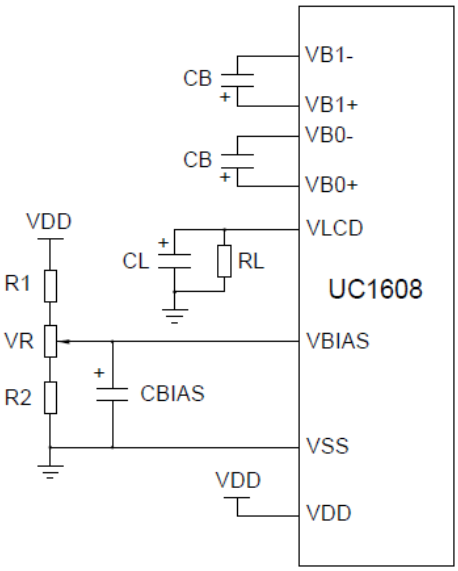
GLCD의 커맨드 리스트.

3.8 Power Supply for Logic and LCD Driving

A. Use internal bias source

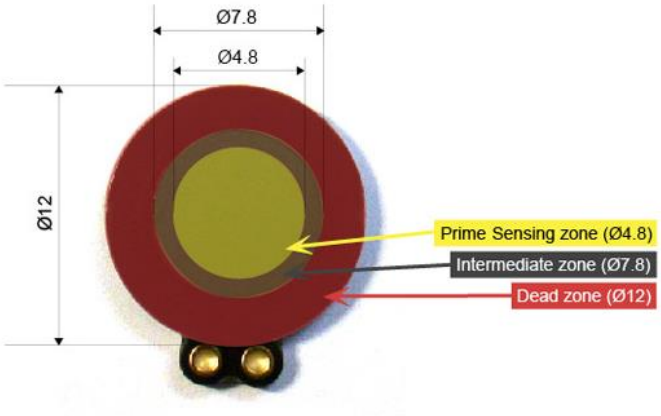


B. Use exernal bias source



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)

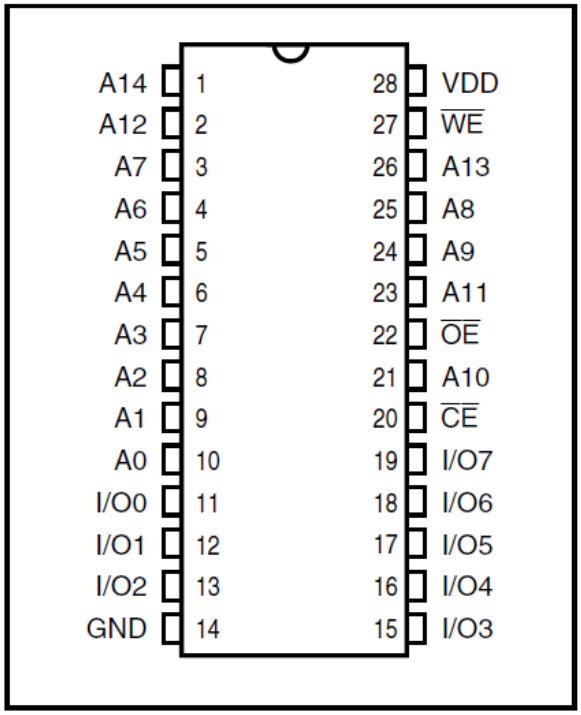
Vin 과 ADC를 연결해줘야 동작하며,
센서 기판에서 Vin은 'V'로 표기되어 있고, ADC는 'A'로 표기

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.

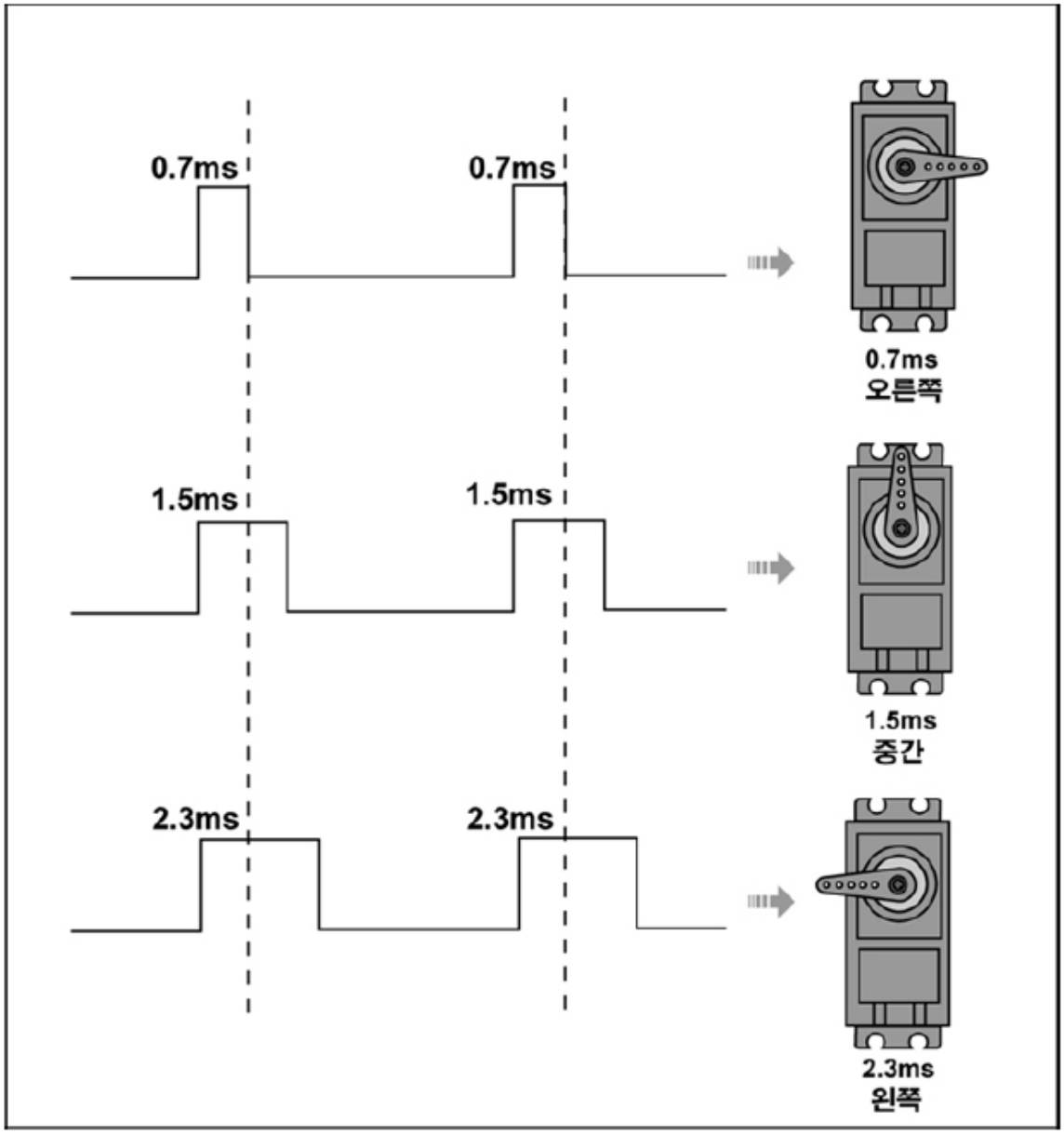
SRAM

PIN CONFIGURATION
28-Pin SOP

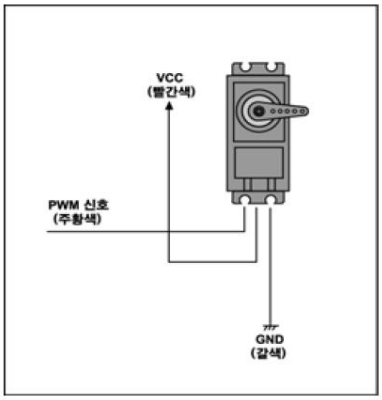


전형적인 SRAM이다. 15비트 주소 입력과 8비트 데이터 입출력 포트를 갖는다. 5V 전원이 공급되어야 하며, 최대 응답 속도는 45ns (22.2MHz) 로 ATMEGA128에 충분히 사용 가능.

서보 모터 HES-288

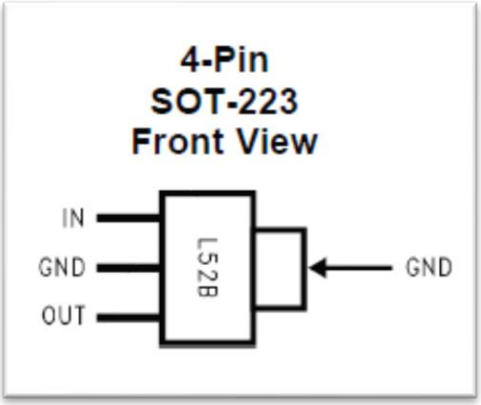


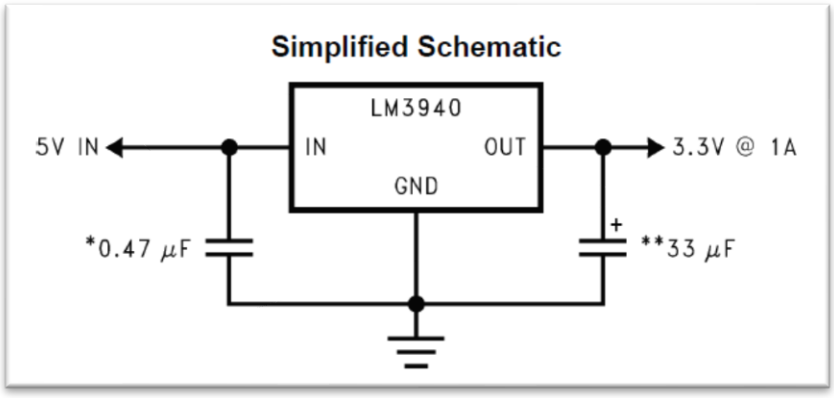
PWM을 이용해 제어되는 서보 모터이다. 게임이 1초에 60번씩 시뮬레이션 되므로, 16.67ms마다 모터에 대해 갱신 신호가 들어간다. OCR1B 비교 매치 인터럽트에서 OC1B 출력을 클리어하는 방법으로, PWM 출력을 소프트웨어 생성



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.

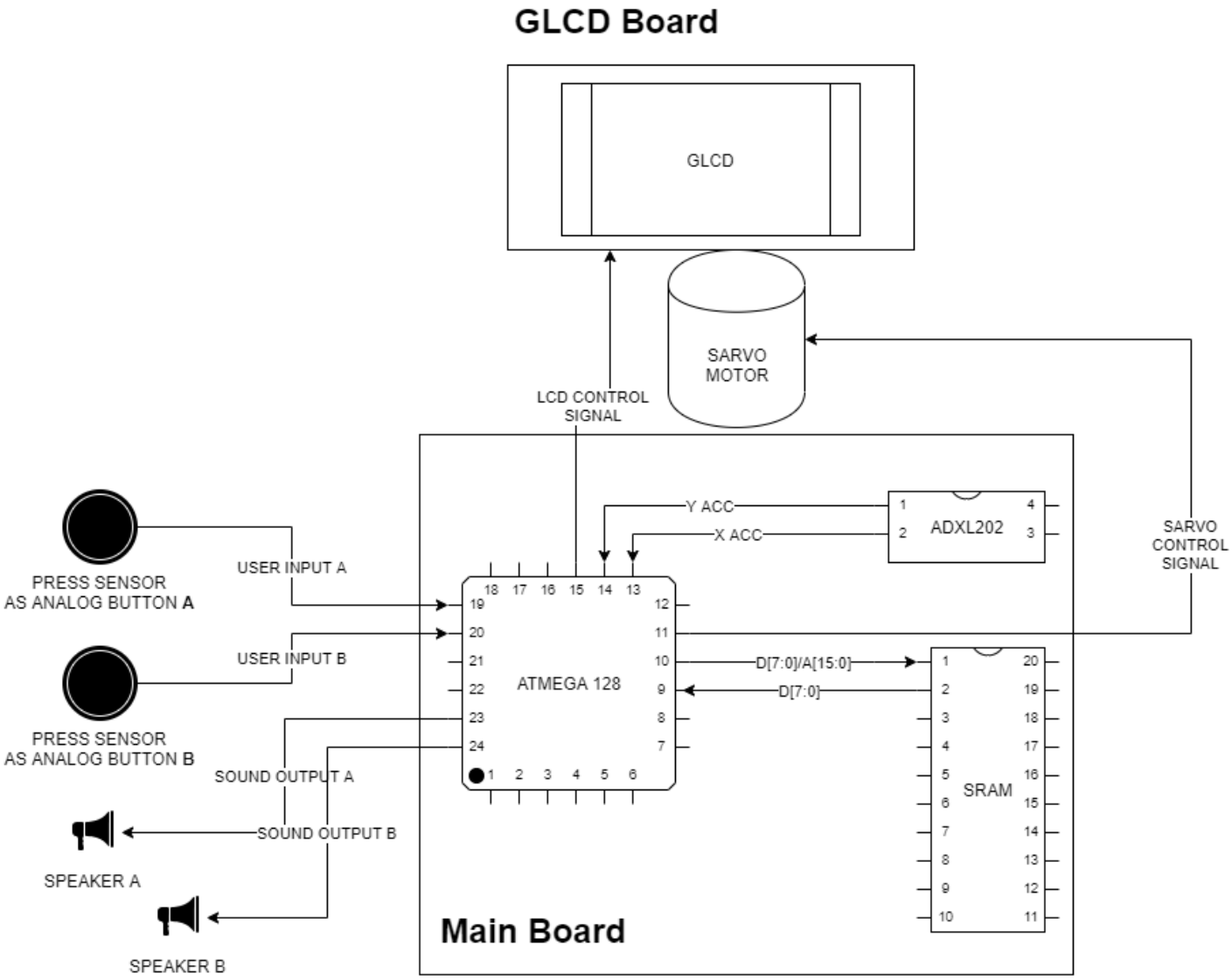




1A의 전류를 출력할 수 있는데, 이 디바이스에 연결되는 LCD의 V_{DD} 소모 전류량은 최대 1.5mA에 불과하므로 크게 문제되진 않는다.

4. 작품 구성도 및 회로도

작품 구성도



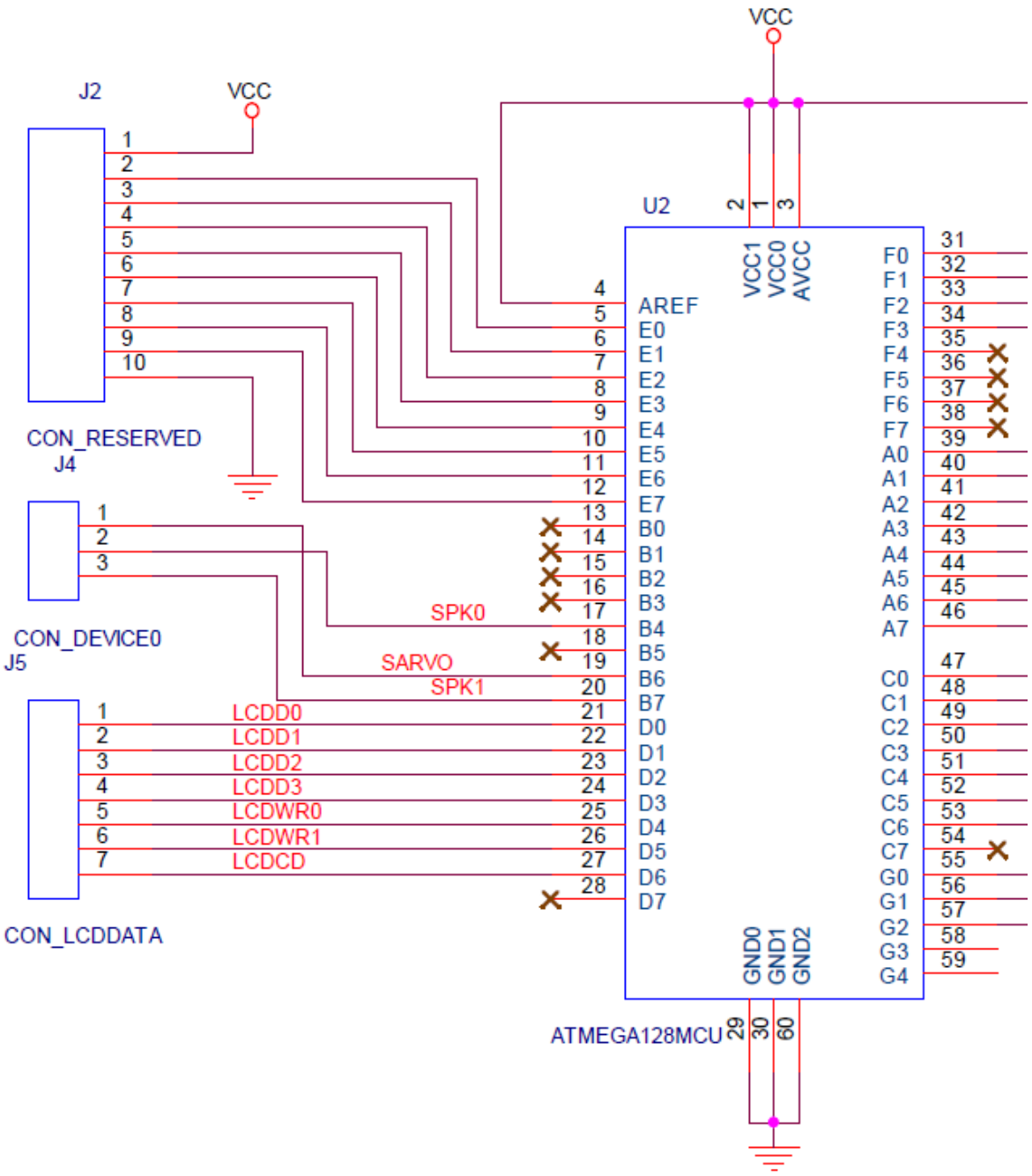


Figure 1. 외부 커넥터 연결

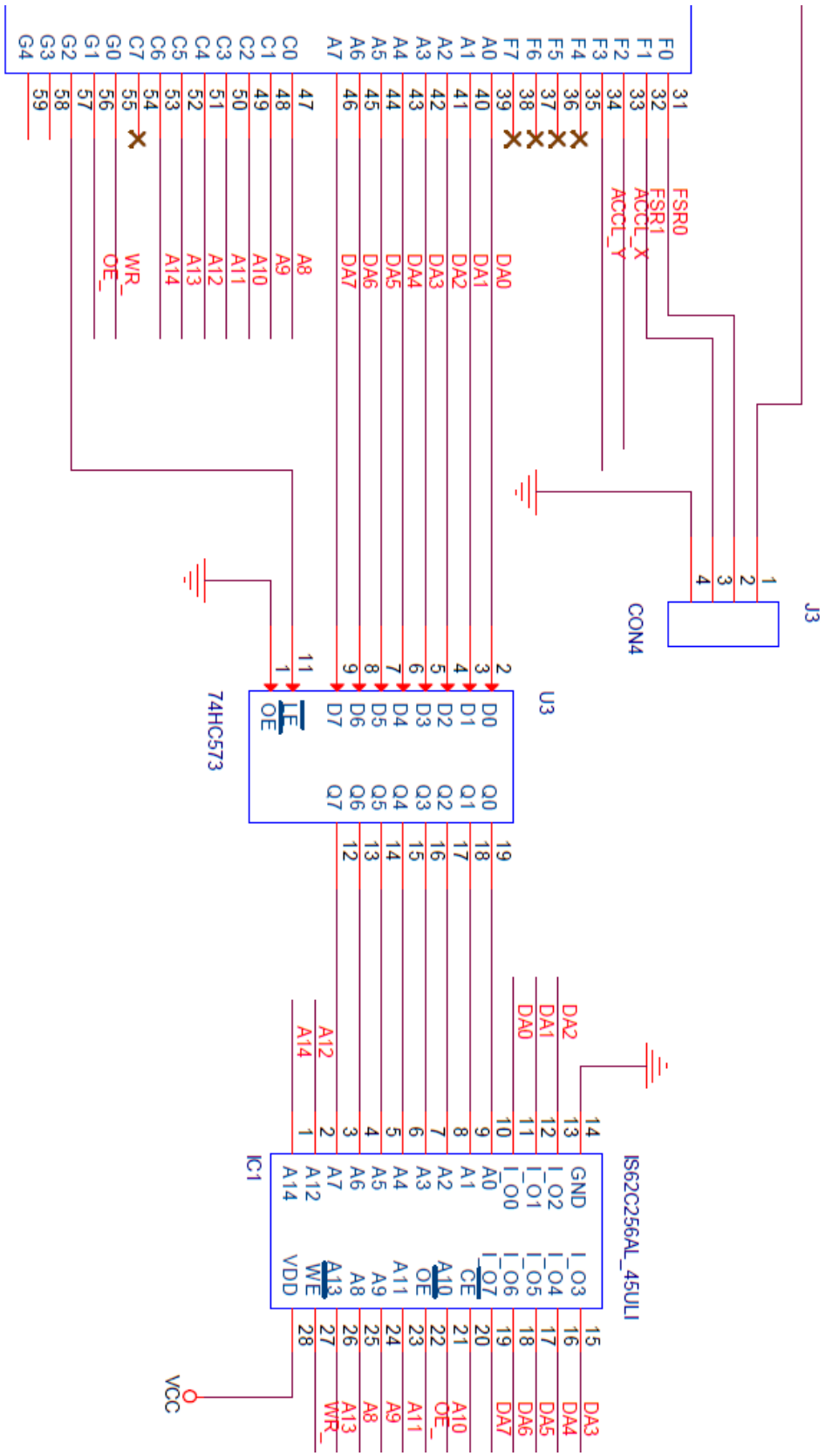


Figure 2. 메모리 연결

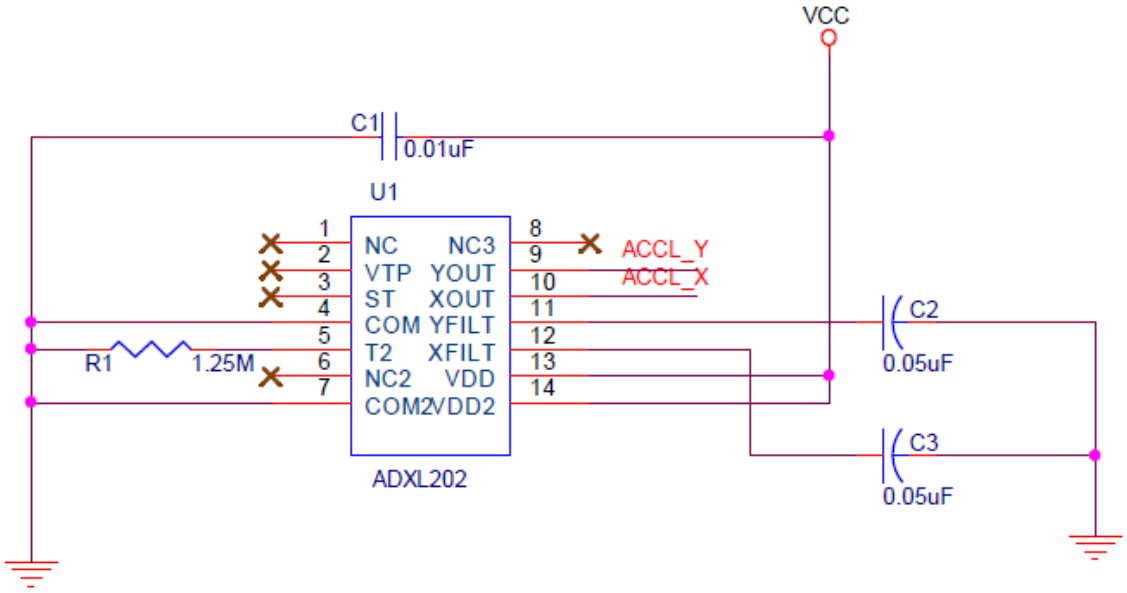


Figure 3. 가속도 센서

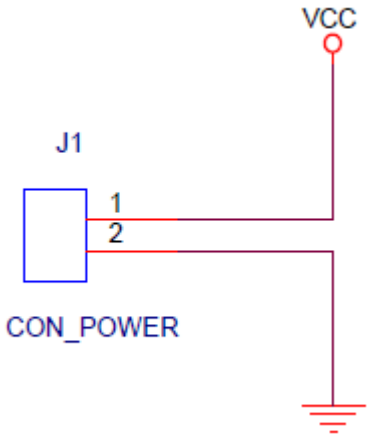


Figure 4. 전원 커넥터

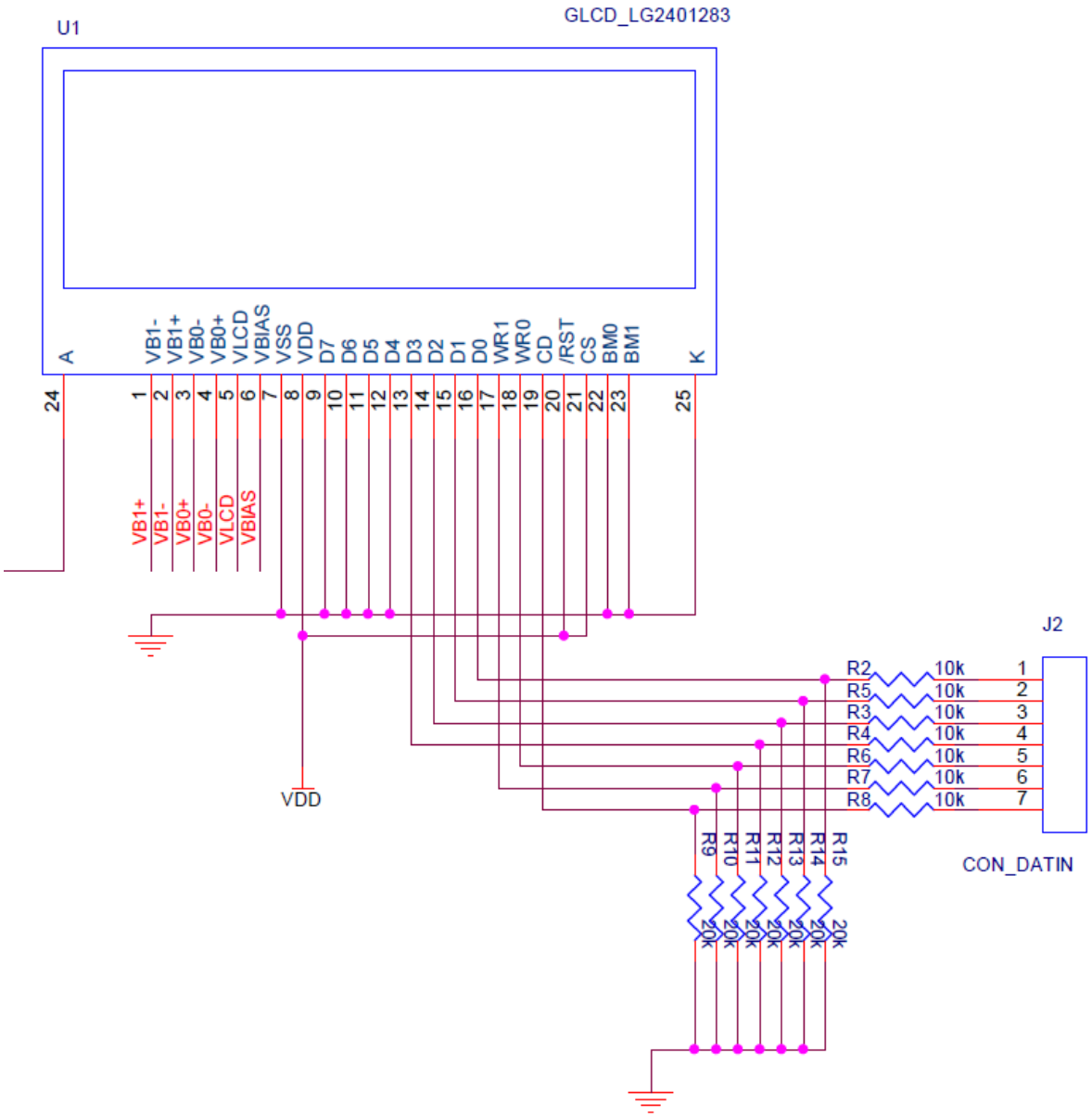


Figure 5. 커넥터 연결.

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

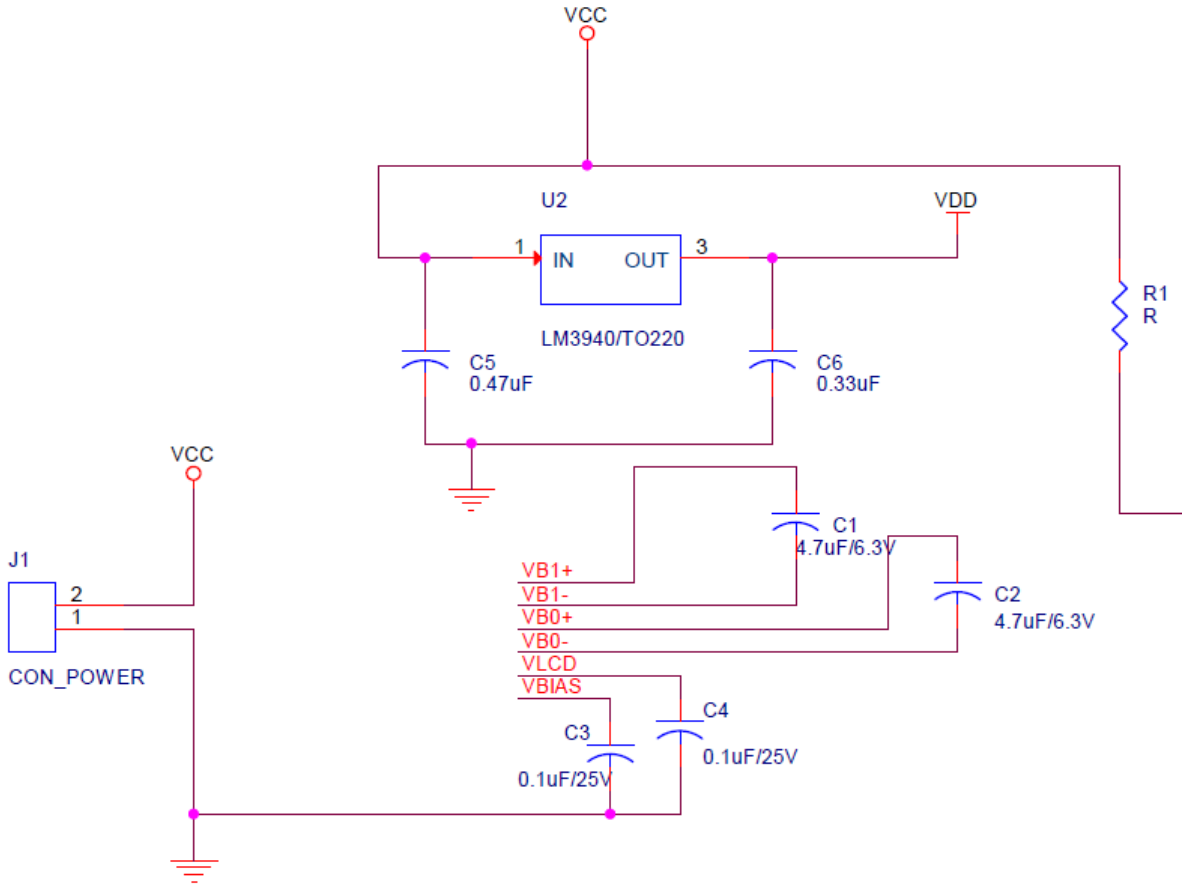


Figure 6. 바이어스

5. 작품 진행상황

첫 주에 PCB 기판의 설계를 잘못 해서 새로운 기판을 뽑아야 하는 관계로, 하드웨어 부분에서는 아무런 진전이 없었다.

소프트웨어 부분에서는 생산성을 높이기 위해 CodeVision IDE가 아닌 Visual Studio IDE에서, 오픈 소스 AVR 컴파일러인 AVR-GCC와 AVR 프로그래머 유틸리티인 AvrDude를 이용해 AVR-C 개발 환경을 구성하였다.

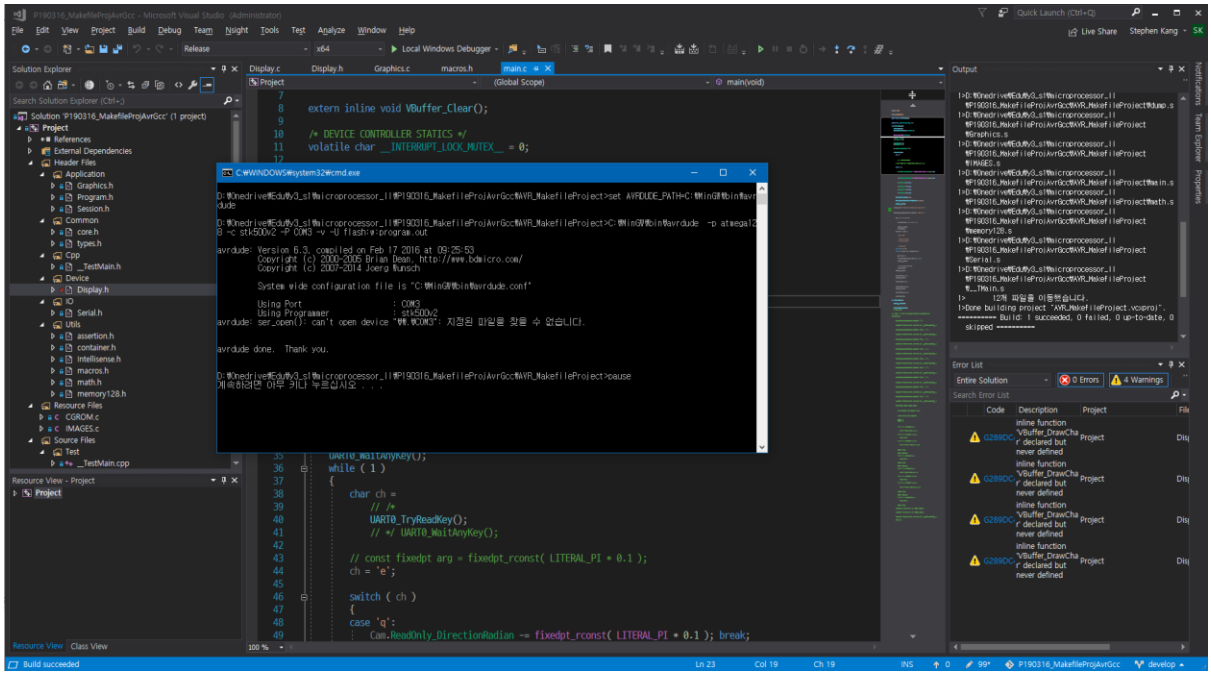


Figure 7. 개발 환경

지금까지 완성된 기능은 대부분 앞으로의 개발을 돕기 위한 기본적인 컨테이너와 유틸리티 기능들이다. 특히, 본격적으로 하드웨어를 완성하고 개발에 착수하기 전, 여러 가지 복잡한 함수성을 테스트하기 위해 시리얼 포트를 이용한 여러 가지 디버깅 함수에 공을 들였다. 아래는 *assertion.h* 파일에 선언된 매크로 및 함수들이다.

```
#ifndef _DEBUG
#define assertf(...)
#define verifyf(expr, ...) if(!is_true(expr)) { while(1); }
#define log_verbose(...)
#define log_display(...) { char __buff__[256]; sprintf(__buff__, __VA_ARGS__);
outputmsg_uart0(__buff__); outputmsg_uart0("\r\n"); }
#else // defined _DEBUG
#if LOG_VERBOSE
#define log_verbose(...) log_display(__VA_ARGS__)
#else
#define log_verbose(...)
#endif

#define log_display(...) { char __buff__[256]; sprintf(__buff__, __VA_ARGS__);
```

```
internal_logslow(__FILE__, __LINE__, __buff__ );}

#define verifyf(expr, ...) assertf(expr, __VA_ARGS__)
#define assertf(expr, ...) if(!is_true(expr)) { char __buff__[256]; sprintf(__buff__,
__VA_ARGS__); internal_assertion_failed(__FILE__, __LINE__, __buff__ );}

/** Triggered when the assertion has failed.
 * To debug the arguments, dumps memory and output to rs232 serial port.
 */
void internal_assertion_failed( const char* file, int line, const char* msg );
void internal_logslow( const char* file, int line, const char* buff );
#endif

// Send message synchronously.
void outputmsg_uart0( const char* msg );
```

어서트 함수는 아래와 같이 사용된다. 첫 번째 조건식이 거짓인 경우, 어서트 문자열이 시리얼을 통해 출력된다. 하드웨어가 완성된 이후에는 디스플레이에 출력될 예정이다.

```
assertf( xCol != NULL && y != NULL, "Input index must not be null!" );
```

특히, 많은 부분이 조건부 매크로 블록 안에 들어가 있는데, 릴리즈 시점에 *macro.h* 파일의 여러 플래그를 조정해 이런 디버깅 기능 활성화 여부를 결정할 수 있게끔 하였다.

아래는 여러 가지 플래그와, 프로그램 전체에서 필수적으로 사용되는 유틸리티인 마스킹 매크로가 정의되어 있는 *macros.h* 파일이다.

```
#define _MASK(num) (0x01 <<(num))
#define _MASK_SETBIT0(_a) (_MASK(_a))
#define _MASK_SETBIT1(_a, _b) (_MASK(_a)|_MASK(_b))
#define _MASK_SETBIT2(_a, _b, _c) (_MASK(_a)|_MASK(_b)|_MASK(_c))
#define _MASK_SETBIT3(_a, _b, _c, _d) (_MASK(_a)|_MASK(_b)|_MASK(_c)|_MASK(_d))
#define _MASK_SETBIT4(_a, _b, _c, _d, _e)
(_MASK(_a)|_MASK(_b)|_MASK(_c)|_MASK(_d)|_MASK(_e))
#define _MASK_SETBIT5(_a, _b, _c, _d, _e, _f)
(_MASK(_a)|_MASK(_b)|_MASK(_c)|_MASK(_d)|_MASK(_e)|_MASK(_f))
#define _MASK_SETBIT6(_a, _b, _c, _d, _e, _f, _g)
(_MASK(_a)|_MASK(_b)|_MASK(_c)|_MASK(_d)|_MASK(_e)|_MASK(_f)|_MASK(_g))
#define _MASK_SETBIT7(_a, _b, _c, _d, _e, _f, _g, _h)
(_MASK(_a)|_MASK(_b)|_MASK(_c)|_MASK(_d)|_MASK(_e)|_MASK(_f)|_MASK(_g)|_MASK(_h))
#define _MASK_GETMACRO(_0, _1, _2, _3, _4, _5, _6, _7, NAME, ...) NAME
#define mask(...) _MASK_GETMACRO(__VA_ARGS__, _MASK_SETBIT7, _MASK_SETBIT6,
_MASK_SETBIT5, _MASK_SETBIT4, _MASK_SETBIT3, _MASK_SETBIT2, _MASK_SETBIT1,
_MASK_SETBIT0)(__VA_ARGS__)

#define ARRAYCOUNT(Array) (sizeof(Array)/sizeof(*Array))

extern volatile char __INTERRUPT_LOCK_MUTEX__;
#define INTERRUPT_LOCK_MUTEX (*((volatile byte*)&__INTERRUPT_LOCK_MUTEX__))

#define DISABLE_INTERRUPT cli(); ++INTERRUPT_LOCK_MUTEX;
#define ENABLE_INTERRUPT --INTERRUPT_LOCK_MUTEX; if(INTERRUPT_LOCK_MUTEX <= 0)
{ sei(); }

#define portc_dbgout(val) PORTC = val

#define ENABLE_DEBUG 1

#if ENABLE_DEBUG
```

```
#define LOG_VERBOSE 0
#define LOG_MEMORY 0
#endif

#define ENABLE_HEAP_CACHE 1
#define USE_STDLIB_MALLOC 1
#define USE_SERIAL_COMMUNICATION 1

#if ENABLE_DEBUG
#ifndef _DEBUG
#define _DEBUG
#endif
#endif
```

특히, DISABLE_INTERRUPT와 ENABLE_INTERRUPT 매크로는 원자성이 보장되어야 하는 함수들 내부에서 일시적으로 인터럽트를 비활성화 시키기 위해 사용되었다. 내부적으로는 Semaphore를 사용해, Diable된 횟수가 0인 경우에만 인터럽트가 다시 활성화된다.

아래는 시리얼 포트로 메시지 등을 출력하기 위한 함수이다.

```
void outputmsg_uart0( const char* msg )
{
    DISABLE_INTERRUPT;

    const char* head = msg;
    while ( *head != '\0' )
    {
        while ( !( UCSR0A & 0x20 ) );
        UDR0 = *( head++ );
    }

    while ( !( UCSR0A & 0x20 ) );
    ENABLE_INTERRUPT;
}
```

다음은 여러 데이터를 보관하는 데 사용되는 컨테이너이다. 특히 노드로 연결되는 Linked List는 프로그램 전반에 걸쳐 대단히 핵심적인 역할을 수행하므로, 개발 전반에 걸쳐 최적화를 다듬을 예정이다. 아래에는 지면상 헤더 파일만을 포함하였다.

```
#pragma once
#include "Intellisense.h"
#include "types.h"
#include "assertion.h"
#include <string.h>
#include "memory128.h"

/** Dynamic array. All struct elements are read only. */
// N must be 8 * byte size
typedef struct TArray {
    uint16 _count;
    uint8 _ofst;
    uint8* _data;
} TArray;

typedef TArray FString;
```

```
typedef struct TListNode {
    struct TListNode* Prev;
    struct TListNode* Next;
    void* Element;
} TListNode;

typedef struct TList {
    uint8 _ofst;
    // CAUTION: READ ONLY!
    TListNode* Head;
    TListNode* Tail;
} TList;

inline void TArray_Initialize( TArray* const pArray, const uint8 ElementSize, const
size_type Capacity );
inline void TArray_Dispose( TArray* const pArray );
/* returns element pointer.    */
inline void* TArray_At( TArray* const pArray, size_type Index );
/* returns new element index. */
size_type TArray_AddLast( TArray* const pArray, void const* const Element );
void TArray_RemoveElement( TArray* const pArray, size_type Index );

void FString_Initialize( FString* const pString, const char* InitData );


inline void TList_Initialize( TList* const pList, const uint8 ElementSize );
void TList_Dispose( TList* const pList );
void TList_PushFront( TList* const pList, const void* const Element );
void TList_PushBack( TList* const pList, const void* const Element );
void TList_Insert( TList* const pList, TListNode* const pPrecededNode, const void*
const Value );
void TList_PopFront( TList* const pList );
void TList_PopBack( TList* const pList );
void TListNode_Remove( TListNode* const pNode );

inline void* TArray_At( TArray* const pArray, size_type Index )
{
    uint8* pCursor = pArray->_data + Index * pArray->_ofst;
    assertf( (void*) pCursor < GetMemoryBound( pArray->_data ), "Invalid memory
access!" );
    return pCursor;
}

inline void TArray_Initialize( TArray* const pArray, const uint8 ElementSize, const
size_type Capacity )
{
    pArray->_ofst = ElementSize;
    pArray->_data = (uint8*) Malloc( Capacity * pArray->_ofst );
    pArray->_count = 0;
}

inline void TList_Initialize( TList* const pList, const uint8 ElementSize )
{
    pList->_ofst = ElementSize;
    pList->Head = pList->Tail = NULL;
}

inline void TArray_Dispose( TArray* const pArray )
{
    Free( pArray->_data );
```

```
}
```

다음은 Rasterization과 디바이스로의 출력을 맡는 디스플레이 관련 함수성이다. 프로그램은 내부적으로 [디스플레이 폭] * [디스플레이 높이] / 8 byte 크기의 비디오 버퍼를 갖게 되며, 일정 주기마다 버퍼 전체를 디스플레이로 출력하는 함수를 호출하게 된다.

```
#define PIXELS_PER_BYTE 8
#define LCD_WIDTH 72 // Must be multiplicand of 8
#define LCD_HEGIHT 23
#define LCD_LINE_BYTE (LCD_WIDTH / PIXELS_PER_BYTE)

#define LCD_BUFFER_LENGTH (LCD_WIDTH * LCD_HEGIHT / PIXELS_PER_BYTE)

extern byte* LCDBuffer;
enum { CGROM_CHARACTER_BYTE_SIZE = 16 };
enum { CGROM_TRUNC_BEGIN = 1 };
enum { CGROM_TRUNC_END = 3 };
enum { CGROM_DISPLAY_HEIGHT = CGROM_CHARACTER_BYTE_SIZE - CGROM_TRUNC_BEGIN -
CGROM_TRUNC_END };
extern const char CGROM[2048];

void LCDDevice__Initialize();
void LCDDevice__Render();

void VBuffer_Clear();

inline void VBuffer_DrawDot( int16 x, int16 y )
{
    if( 0 <= x && x < LCD_WIDTH
        && 0 <= y && y < LCD_HEGIHT )
    {
        const byte Page = x >> 3;
        const byte Idx = x & 0b111;
        // @todo. Apply pixel filter by using masking buffer.
        const uint16 Block = LCD_LINE_BYTE * y + Page;
        const byte Mask = mask( 7 - Idx );
        LCDBuffer[Block] |= Mask;
    }
}

inline 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 );
```

점 그리기, 선 그리기, 글자 출력 등의 오퍼레이션을 제공한다. 한 비트가 점 하나를 나타내므로 비트연산을 사용하게 된다. 추후 마스킹 버퍼 등을 사용해, 화면의 특정 부분에는 픽셀을 그릴 수 없게끔 만들 예정이다. (UI 등에 사용)

직선 그리기에는 브레젠험의 선 그리기 알고리즘이 사용되었다.

```
void VBuffer_DrawLine( int16 x0, int16 y0, const int16 x1, const int16 y1 )
{
    // @todo. line verification
    int16 dx = math_abs( x1 - x0 );
    int16 dy = -math_abs( y1 - y0 );
    int8 sx = x1 > x0 ? 1 : -1;
    int8 sy = y1 > y0 ? 1 : -1;
    int32 err = dx + dy, e2;

    while ( 1 )
    {
        VBuffer_DrawDot( x0, y0 );
        if ( x0 == x1 && y0 == y1 ) { break; }
        e2 = err * 2;
        if ( e2 >= dy )
        {
            err += dy;
            x0 += sx;
        }
        if ( e2 <= dx )
        {
            err += dx;
            y0 += sy;
        }
    }
}
```

마지막으로, 3D 그래픽을 흉내내는 그래픽스 클래스이다. 게임 월드는 Z 좌표가 없는 2D 로 구현되지만, 렌더링 될 때에는 X 를 Forward, Y 를 Rightward, Z 를 Upward 로 하여 3D 처럼 이미지를 그려낸다.

아래는 오브젝트의 위치를 나타내는 데 사용되는 32 비트 정수 벡터이다.

```
typedef struct FPoint16 {
    int16 x;
    int16 y;
} FPoint16;
```

또한 모든 볼 수 있는 물체는 선의 집합으로 나타나게 되는데, 즉 거리에 따라 크기는 달라지지만 방향은 항상 카메라 방향을 보고 있게 되는, 일종의 빌보드 효과를 내게 된다.

```
typedef struct FLineInfo {
    FPoint8 Begin;
    FPoint8 End;
} FLineInfo;

typedef struct FLineVector {
    FLineInfo const * Lines;
    uint8 NumLines;
  } FLineVector;

 const FLineInfo src_triangle[] = {
    -1, -1, 0, 1, /**/ 0, 1, 1, -1, /**/ 1, -1, -1, -1
};
```

다음은 카메라의 위치를 나타내는 구조체로, 다른 오브젝트와 마찬가지로 위치를 갖게 되며 현재 방향을 라디안으로 나타낸다.

```
typedef struct FCameraTransform {
    FPoint16 Position;
    // DirectionRadian is -pi ~ +pi
    fixedpt ReadOnly_DirectionRadian;
    // Transform cache should be refreshed on every rendering request.
    FPointFP CachedDirection;
} FCameraTransform;
```

즉, 물체가 렌더링 되는 과정을 요약하면 다음과 같다. 카메라의 위치와 오브젝트의 위치 사이의 거리를 이용해 원근감을 계산하고 카메라의 방향 라디안을 이용해 카메라의 방향 벡터를 구한 뒤, 카메라의 방향 벡터와 카메라에서 그릴 대상 오브젝트를 향하는 방향 벡터 사이의 각도를 찾는다.

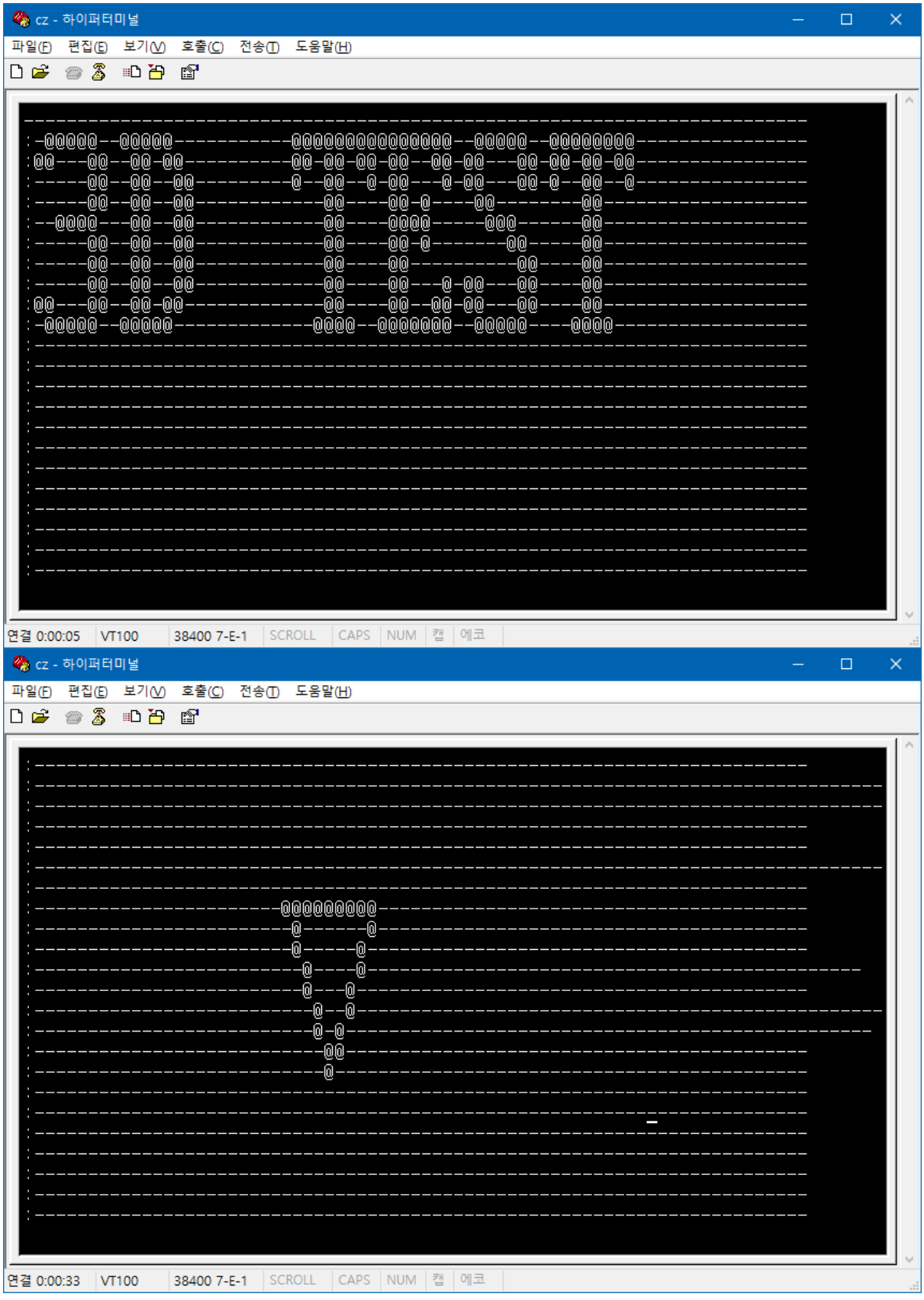
```
 inline bool CalculateAngleIfVIsible( const FPoint16* Position, const
FCameraTransform* Camera, int8* DegreesWhenVisible, int16* Distance )
{
    FPointFP DirectionVector, CameraDirectionUnitVector;
    fixedpt AngleBetween;
    fixedpt DistanceFromCamera;
    fixedpt Z;

    DirectionVector.x = fixedpt_fromint( Position->x - Camera->Position.x );
    DirectionVector.y = fixedpt_fromint( Position->y - Camera->Position.y );
    CameraDirectionUnitVector = Camera->CachedDirection;

    DistanceFromCamera = sz( DirectionVector );
    if ( DistanceFromCamera > fixedpt_rconst( MINIMAL_VISIBLE_DISTANCE ) )
    {
#if LOG_VERBOSE
        *Distance = fixedpt_toint( DistanceFromCamera );
#endif
        return false;
    }

    // acos(dot(a,b) / (sz(a)*sz(b)))
    AngleBetween = fixedpt_div( dot( DirectionVector, CameraDirectionUnitVector ),
DistanceFromCamera );
    AngleBetween = fixedpt_acos( AngleBetween );
    Z = fixedpt_mul( CameraDirectionUnitVector.x, DirectionVector.y ) -
fixedpt_mul( CameraDirectionUnitVector.y, DirectionVector.x );
    *DegreesWhenVisible = fixedpt_toint( fixedpt_div( fixedpt_mul( Z > 0 ?
AngleBetween : -AngleBetween, fixedpt_rconst( 180.0 ) ), FIXEDPT_PI ) );
    *Distance = fixedpt_toint( DistanceFromCamera );

    const fixedpt AngleLimit = fixedpt_rconst( CAMERA_FOV * LITERAL_PI / 180.0 );

    return AngleBetween < AngleLimit;
}
```

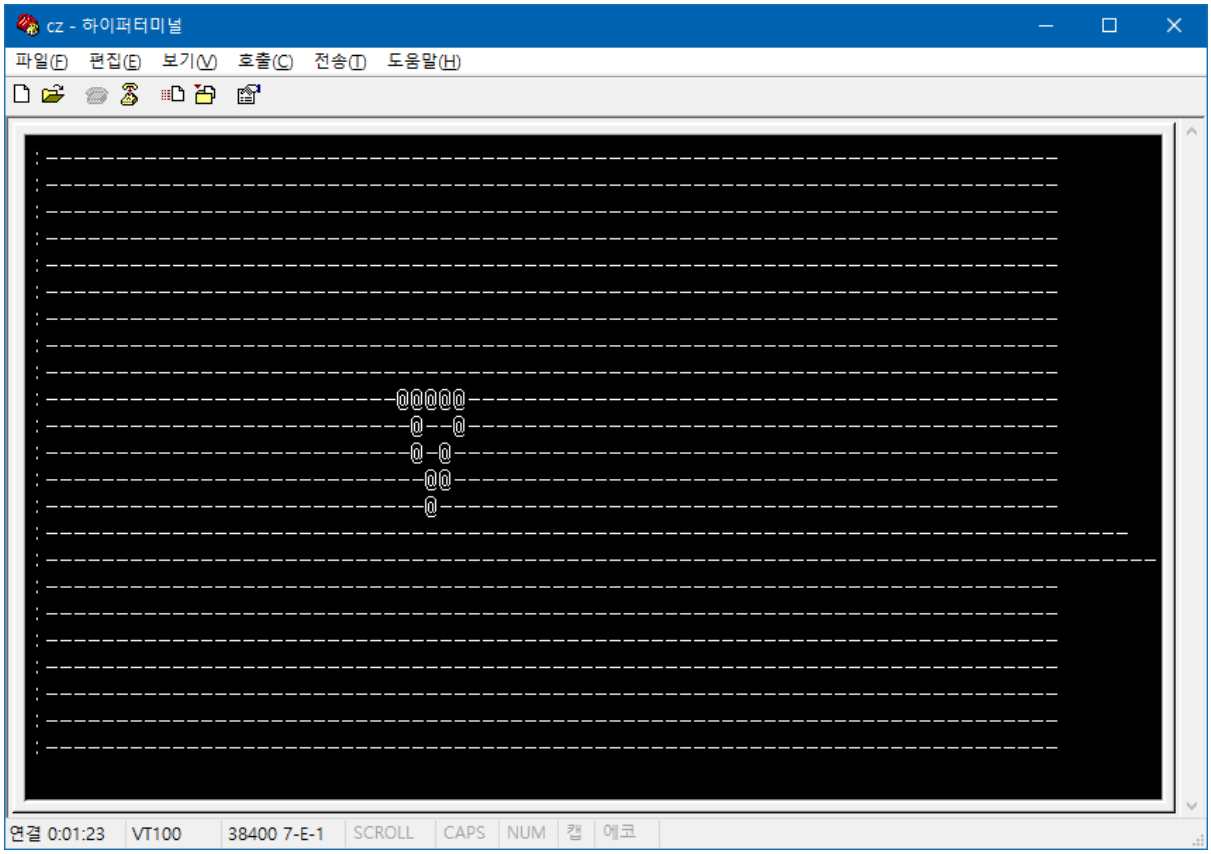
정확한 계산을 위해 고정 소수점을 사용한다. 고정 소수점 라이브러리는 오픈 소스로 공개된 것을 참고하였다.

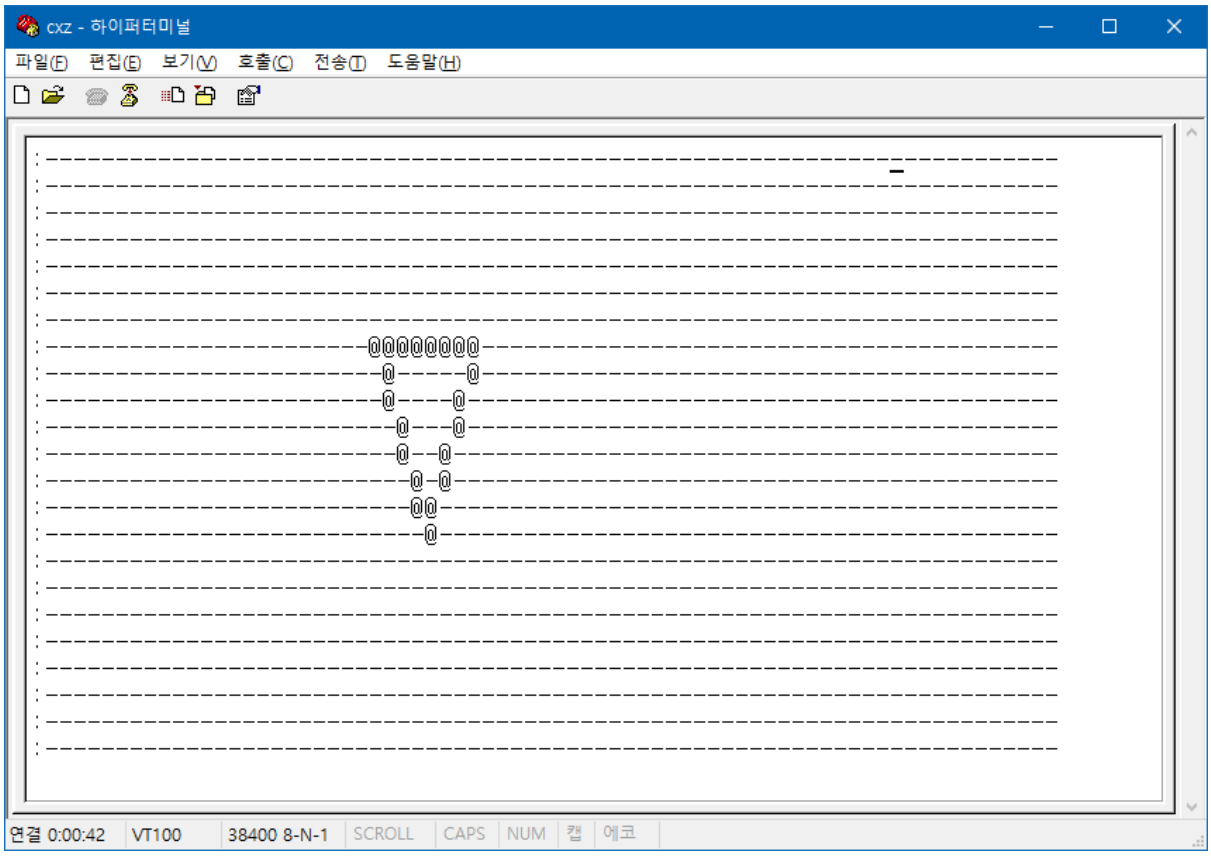
아래는 지금까지 서술한 기능들을 종합, 기본적인 렌더링 성능을 시험하기 위해 작성된 테스트 코드이다.

```
void main( void )
{
    InitializeDevice();
    CSerialSender_Initialize( &UART0Sender );
    // Setup draw args
    DECLARE_LINE_VECTOR( Triangle );
    CDrawArgs Arg;
    Arg.Mesh = Triangle;
    Arg.Position.x = 50;
    Arg.Position.y = 0;
    // Setup camera
    FCameraTransform Cam;
    Cam.Position.x = 0;
    Cam.Position.y = 0;
    Cam.ReadOnly_DirectionRadian = 0;
    CalculateTranformCache( &Cam );

    UART0_WaitAnyKey();
    {
        byte x = 0, y = 0;
        VBuffer_DrawString( &x, &y, "3D TEST", false );
        LCDDevice__Render();
    }
    UART0_WaitAnyKey();
    while ( 1 )
    {
        char ch =
            // /*
            UART0_TryReadKey();
            // */ UART0_WaitAnyKey();

        // const fixedpt arg = fixedpt_rconst( LITERAL_PI * 0.1 );

        switch ( ch )
        {
        case 'q':
            Cam.ReadOnly_DirectionRadian -= fixedpt_rconst( LITERAL_PI * 0.01 ); break;
        case 'e':
            Cam.ReadOnly_DirectionRadian += fixedpt_rconst( LITERAL_PI * 0.01 ); break;
        case 'w':
            Cam.Position.x += 5; break;
        case 's':
            Cam.Position.x -= 5; break;
        case 'a':
            Cam.Position.y -= 5; break;
        case 'd':
            Cam.Position.y += 5; break;
        }

        VBuffer_Clear();
        CalculateTranformCache( &Cam );
        CDrawArgs_DrawOnDisplayBufferPerspective( &Arg, &Cam );
        LCDDevice__Render();
    }
}
```

다음은 하이퍼터미널을 이용, 위의 코드를 테스트한 결과이다. 이는 시리얼 통신을 이용, 버퍼를 ASCII로 변환해 내보낸 것으로 실제로는 글자 하나가 비트 하나에 대응되는 것과 같다.




cz - 하이퍼터미널
파일(F) 편집(E) 보기(V) 호출(C) 전송(T) 도움말(H)
연결 0:01:23 VT100 38400 7-E-1 SCROLL CAPS NUM 캡 에코


cxz - 하이퍼터미널
파일(F) 편집(E) 보기(V) 호출(C) 전송(T) 도움말(H)
연결 0:00:42 VT100 38400 8-N-1 SCROLL CAPS NUM 캡 에코