





강사 – Innova Lee(이상훈)
gcccompil3r@gmail.com
학생 – 정한별
hanbulkr@gmail.com

I2C (Lidar-lite v3) 프로그래밍

2018. 8. 7. 21:49

<https://blog.naver.com/hanbulkr/221334501552>

-  SPI_LIDAR.pdf
-  LIDAR_Lite_v3_Operation_Manual_and_Technical_Specifications.pdf

I2C를 이용해서 Lidar를 돌리는 걸 설명한다.

Garmin 사의 Lidar-lite v3의 datasheet 위에꺼는 안전정보에 대한 긴 글이 있다.

ISPI_LIDAR.pdf

↓

여기가 진짜 manual 이다.


LIDAR_Lite_v3_Operation_Manual_and_Technical_Specifications...

↓

<Lidar 스펙>

Connections

Wiring Harness



Wire Color	Function
Red	5 Vdc (+)
Orange	Power enable (internal pull-up)
Yellow	Mode control
Green	I2C SCL
Blue	I2C SDA
Black	Ground (-)

There are two basic configurations for this device:

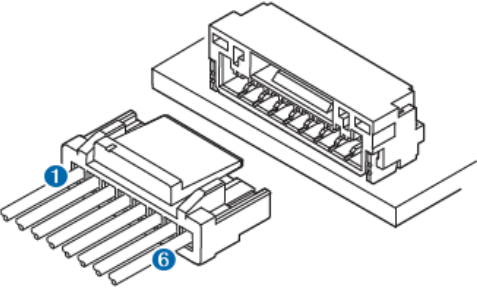
- **I2C (Inter-Integrated Circuit)**—a serial computer bus used to communicate between this device and a microcontroller, such as an Arduino board ("I2C Interface", page 4).
- **PWM (Pulse Width Modulation)**—a bi-directional signal transfer method that triggers acquisitions and returns distance measurements using the mode-control pin ("Mode Control Pin", page 4).

Connector

You can create your own wiring harness if needed for your project or application. The needed components are readily available from many suppliers.

Part	Description	Manufacturer	Part Number
Connector housing	6-position, rectangular housing, latch-lock connector receptacle with a 1.25 mm (0.049 in.) pitch.	JST	GHR-06V-S
Connector terminal	26-30 AWG crimp socket connector terminal (up to 6)	JST	SSHL-002T-P0.2
Wire	UL 1061 26 AWG stranded copper	N/A	N/A

Connector Port Identification



Item	Pin	Function
1	1	5 Vdc (+)
	2	Power enable (internal pull-up)
	3	Mode control
	4	I2C SCL
	5	I2C SDA
6	6	Ground (-)

Interface

Specification	Measurement
User interface	I2C PWM External trigger
I2C interface	Fast-mode (400 kbit/s) Default 7-bit address 0x62 Internal register access & control
PWM interface	External trigger input PWM output proportional to distance at 10 μ s/cm

Laser

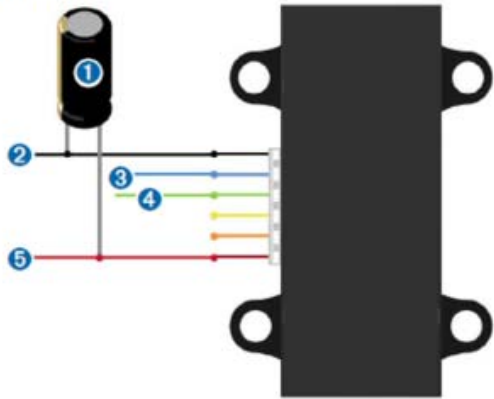
Specification	Measurement
Wavelength	905 nm (nominal)
Total laser power (peak)	1.3 W
Mode of operation	Pulsed (256 pulse max. pulse train)
Pulse width	0.5 μ s (50% duty Cycle)
Pulse train repetition frequency	10-20 KHz nominal
Energy per pulse	<280 nJ
Beam diameter at laser aperture	12 \times 2 mm (0.47 \times 0.08 in.)
Divergence	8 mRadian

<Lidar 배선도>

- 우리는 커패시터를 이용해 기본적인 I2C 모드로 확인 한다.
- 저항을 연결한 경우 PWM 모드로 동작하는 것을 볼 수 있다.

I2C Connection Diagrams

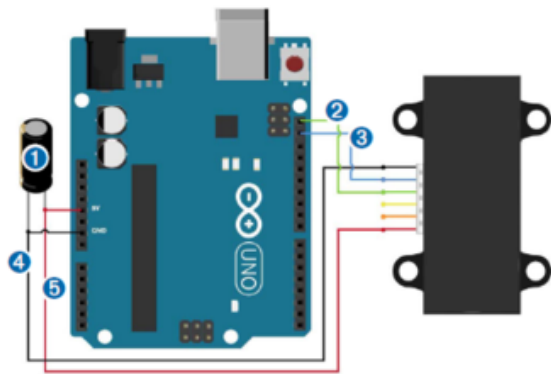
Standard I2C Wiring



Item	Description	Notes
1	680µF electrolytic capacitor	You must observe the correct polarity when installing the capacitor.
2	Power ground (-) connection	Black wire
3	I2C SDA connection	Blue wire
4	I2C SCA connection	Green wire
5	5 Vdc power (+) connection	Red wire The sensor operates at 4.75 through 5.5 Vdc, with a max. of 6 Vdc.

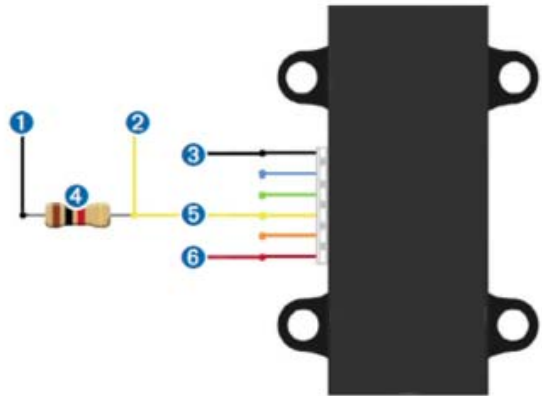
Standard Arduino I2C Wiring

Standard Arduino I2C Wiring



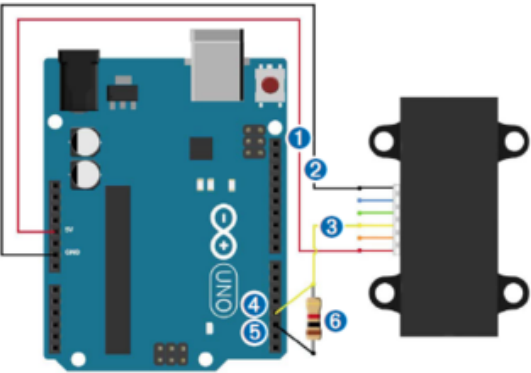
Item	Description	Notes
1	680µF electrolytic capacitor	You must observe the correct polarity when installing the capacitor.
2	I2C SCA connection	Green wire
3	I2C SDA connection	Blue wire
4	Power ground (-) connection	Black wire
5	5 Vdc power (+) connection	Red wire The sensor operates at 4.75 through 5.5 Vdc, with a max. of 6 Vdc.

PWM Wiring



Item	Description	Notes
1	Trigger pin on microcontroller	Connect the other side of the resistor to the trigger pin on your microcontroller.
2	Monitor pin on microcontroller	Connect one side of the resistor to the mode-control connection on the device, and to a monitoring pin on your microcontroller.
3	Power ground (-) connection	Black Wire
4	1kΩ resistor	
5	Mode-control connection	Yellow wire
6	5 Vdc power (+) connection	Red wire The sensor operates at 4.75 through 5.5 Vdc, with a max. of 6 Vdc.

PWM Arduino Wiring



Item	Description	Notes
1	5 Vdc power (+) connection	Red wire The sensor operates at 4.75 through 5.5 Vdc, with a max. of 6 Vdc.
2	Power ground (-) connection	Black Wire
3	Mode-control connection	Yellow wire
4	Monitor pin on microcontroller	Connect one side of the resistor to the mode-control connection on the device, and to a monitoring pin on your microcontroller.
5	Trigger pin on microcontroller	Connect the other side of the resistor to the trigger pin on your microcontroller.
6	1kΩ resistor	

<Control 레지스터 리스트>

- 밑에 있는 레지스터를 통해 어떤 명령어들이 가능한지 대략적으로 유추가 가능하다.

Register Definitions

Control Register List

Address	R/W	Name	Description	Initial Value	Details
0x00	W	ACQ_COMMAND	Device command	--	page 8
0x01	R	STATUS	System status	--	page 8
0x02	R/W	SIG_COUNT_VAL	Maximum acquisition count	0x80	page 8
0x04	R/W	ACQ_CONFIG_REG	Acquisition mode control	0x08	page 8
0x09	R	VELOCITY	Velocity measurement output	--	page 8
0x0c	R	PEAK_CORR	Peak value in correlation record	--	page 8
0x0d	R	NOISE_PEAK	Correlation record noise floor	--	page 8
0x0e	R	SIGNAL_STRENGTH	Received signal strength	--	page 9
0x0f	R	FULL_DELAY_HIGH	Distance measurement high byte	--	page 9
0x10	R	FULL_DELAY_LOW	Distance measurement low byte	--	page 9
0x11	R/W	OUTER_LOOP_COUNT	Burst measurement count control	0x01	page 9
0x12	R/W	REF_COUNT_VAL	Reference acquisition count	0x05	page 9
0x14	R	LAST_DELAY_HIGH	Previous distance measurement high byte	--	page 9
0x15	R	LAST_DELAY_LOW	Previous distance measurement low byte	--	page 9
0x16	R	UNIT_ID_HIGH	Serial number high byte	Unique	page 9
0x17	R	UNIT_ID_LOW	Serial number low byte	Unique	page 9
0x18	W	I2C_ID_HIGH	Write serial number high byte for I2C address unlock	--	page 9
0x19	W	I2C_ID_LOW	Write serial number low byte for I2C address unlock	--	page 9
0x1a	R/W	I2C_SEC_ADDR	Write new I2C address after unlock	--	page 9
0x1c	R/W	THRESHOLD_BYPASS	Peak detection threshold bypass	0x00	page 9
0x1e	R/W	I2C_CONFIG	Default address response control	0x00	page 9
0x40	R/W	COMMAND	State command	--	page 10
0x45	R/W	MEASURE_DELAY	Delay between automatic measurements	0x14	page 10
0x4c	R	PEAK_BCK	Second largest peak value in correlation record	--	page 10
0x52	R	CORR_DATA	Correlation record data low byte	--	page 10
0x53	R	CORR_DATA_SIGN	Correlation record data high byte	--	page 10
0x5d	R/W	ACQ_SETTINGS	Correlation record memory bank select	--	page 10
0x65	R/W	POWER_CONTROL	Power state control	0x80	page 10

인터페이스

- 슬레이브 주소 0x62
- 읽기 : 0xC4
- 쓰기 : 0xC5
- 통신 속도: 400kHz

Interface

Initialization

On power-up or reset, the device performs a self-test sequence and initializes all registers with default values. After roughly 22 ms distance measurements can be taken with the I2C interface or the Mode Control Pin.

Power Enable Pin

The enable pin uses an internal pullup resistor, and can be driven low to shut off power to the device.

I2C Interface

This device has a 2-wire, I2C-compatible serial interface (refer to I2C-Bus Specification, Version 2.1, January 2000, available from Philips Semiconductor). It can be connected to an I2C bus as a slave device, under the control of an I2C master device. It supports 400 kHz Fast Mode data transfer.

The I2C bus operates internally at 3.3 Vdc. An internal level shifter allows the bus to run at a maximum of 5 Vdc. Internal 3k ohm pullup resistors ensure this functionality and allow for a simple connection to the I2C host.

The device has a 7-bit slave address with a default value of 0x62. The effective 8-bit I2C address is 0xC4 write and 0xC5 read. The device will not respond to a general call. Support is not provided for 10-bit addressing.

측정 값을 읽어오는 간단한 방법에 대한 설명

Setting the most significant bit of the I2C address byte to one triggers automatic incrementing of the register address with successive reads or writes within an I2C block transfer. This is commonly used to read the two bytes of a 16-bit value within one transfer and is used in the following example.

The simplest method of obtaining measurement results from the I2C interface is as follows:

- 1 Write 0x04 to register 0x00.
- 2 Read register 0x01. Repeat until bit 0 (LSB) goes low.
- 3 Read two bytes from 0x8f (High byte 0x0f then low byte 0x10) to obtain the 16-bit measured distance in centimeters.

A list of all available control registers is available on [page 7](#).

For more information about the I2C protocol, see [I2C Protocol Operation \(page 7\)](#).

control pin을 수정하는 방법 (0x04)

Mode Control Pin

The mode control pin provides a means to trigger acquisitions and return the measured distance via Pulse Width Modulation (PWM) without having to use the I2C interface.

The idle state of the mode control pin is high impedance (High-Z). Pulling the mode control pin low will trigger a single measurement, and the device will respond by driving the line high with a pulse width proportional to the measured distance at 10 μ s/cm. A 1k ohm termination resistance is required to prevent bus contention.

The device drives the mode control pin high at 3.3 Vdc. Diode isolation allows the pin to tolerate a maximum of 5 Vdc.

As shown in the diagram [PWM Arduino Wiring \(page 3\)](#), a simple triggering method uses a 1k ohm resistor in series with a host output pin to pull the mode control pin low to initiate a measurement, and a host input pin connected directly to monitor the low-to-high output pulse width.

If the mode control pin is held low, the acquisition process will repeat indefinitely, producing a variable frequency output proportional to distance.

The mode control pin behavior can be modified with the ACQ_CONFIG_REG (0x04) I2C register as detailed in [0x04 \(page 8\)](#).

Setting

- setting으로 속도, 범위, 민감도를 커스터 마이징 할 수 있다.

- Receiver Bias correction 에서 100회의 연속 측정을 시작할 때 마다 보정모드를 주기적으로 해주어야 한다라는 의미 이다.
(한마디로 100회가 overflow가 일어날때 0x04로 한번 바꾸고 그 후 99회는 0x03으로 돌린다.)

0x00 : 디바이스를 리셋해서 레지스터의 값을 처음 기본값 으로 돌린다.

0x03 : 보정모드 없이 동작을 하는 것이다. (하지만 정확도와 속도가 악영향을 받는다.)

0x04 : 보정모드를 하며 동작하는 것이다.

-Maximum Acquisition Count

최대 획득 수는 장거리에서 또는 낮은 타겟 반사율로 발생하는 상관 관계 레코드 피크 (반사 된 신호에서)를 찾기 위해 디바이스가 수집을 통합하는 횟수를 제한합니다. 최소 측정 속도와 최대 범위를 제어합니다. 단위없는 관계는 대략 다음과 같습니다. 비율 = $1 / n$ 및 범위 = $n^{1/4}$. 여기서 n은 획득 횟수입니다.

// 최대 획득 수. (주소 : 0x02 , 첫 값: 0x08)

/*

* 최대 획득 수가 기본 0x08이다 0:7 bit 설정 가능

* 우리는 50 정도로 해 본다. 0x32

*/

Settings

The device can be configured with alternate parameters for the distance measurement algorithm. This can be used to customize performance by enabling configurations that allow choosing between speed, range and sensitivity. Other useful features are also detailed in this section. See the full register map ([Control Register List \(page 7\)](#)) for additional settings not mentioned here.

Receiver Bias Correction

Address	Name	Description	Initial Value
0x00	ACQ_COMMAND	Device command	--

- Write 0x00: Reset device, all registers return to default values
- Write 0x03: Take distance measurement without receiver bias correction
- Write 0x04: Take distance measurement with receiver bias correction

Faster distance measurements can be performed by omitting the receiver bias correction routine. Measurement accuracy and sensitivity are adversely affected if conditions change (e.g. target distance, device temperature, and optical noise). To achieve good performance at high measurement rates receiver bias correction must be performed periodically. The recommended method is to do so at the beginning of every 100 sequential measurement commands.

Maximum Acquisition Count

Address	Name	Description	Initial Value
0x02	SIG_COUNT_VAL	<u>Maximum acquisition count</u>	0x80

The maximum acquisition count limits the number of times the device will integrate acquisitions to find a correlation record peak (from a returned signal), which occurs at long range or with low target reflectivity. This controls the minimum measurement rate and maximum range. The unit-less relationship is roughly as follows: rate = 1/n and range = n^(1/4), where n is the number of acquisitions.

0x02

R/W	Name	Description	Initial Value
R/W	SIG_COUNT_VAL	Maximum acquisition count	0x80

Bit	Function
7:0	Maximum number of acquisitions during measurement

빠른 종료 모드, 감지 민감도 설정

// 측정 모드를 바꿔 준다. (datasheet를 참고해서 bit를 제어해야 한다.)

/* ACQ_CONFIG_REG

* 0x08+(1:0) 00 : Default PWM mode 10us/cm 단위로 duty

* 01 : Status output mode :디바이스는 Busy 동안 pin active high를 구동 할 것이다.

호스트 장치를 인터럽트하는 데 사용할 수 있습니다.

10 : Fixed delay PWM mode : pin을 low로 설정하고 트리거 측정을 중지합니다.

11 : Oscillator output mode : 공칭 31.25kHz 출력.

디바이스의 실리콘 발진기의 정확도는 일반적으로 공칭의 1 % 이내입니다.

이는 거리 측정에 비례하여 영향을 미치며 보상 계수를 적용하여 측정

할 수 있습니다

0x08+(2) 1 : use acquisition count from REF_COUNT_VAL(0x12)

0x08+(3) 1 : disable measurement quick termination 0: enable measurement quick termination

0x08+(4) 1 : disable filter 0: enable filter(일관성을 위해 8개의 값을 참조)

0x08+(5) 1 : Use delay from MEASURE_DELAY(0x45)참조 0: 기본 버스트 모드 프리러닝모드 사용

0x08+(6) 1 : Disable process during measurement 0: enable

*/

// 감지의 민감도 설정 THRESHOLD_BYPASS

/* -recommended-

* 0x20은 민감도가 높고 어려움이 높다.

- * 0x60은 민감도가 낮고 에러율이 낮다.
- * 우리는 그러면 중간 적당히 민감도가 좋게 하여 한다.
- */

Measurement Quick Termination Detection

Address	Name	Description	Initial Value
0x04	ACQ_CONFIG_REG	Acquisition mode control	0x08

You can enable quick-termination detection by clearing bit 3 in this register. The device will terminate a distance measurement early if it anticipates that the signal peak in the correlation record will reach maximum value. This allows for faster and slightly less accurate operation at strong signal strengths without sacrificing long range performance.

Detection Sensitivity

Address	Name	Description	Initial Value
0x1c	THRESHOLD_BYPASS	Peak detection threshold bypass	0x00

The default valid measurement detection algorithm is based on the peak value, signal strength, and noise in the correlation record. This can be overridden to become a simple threshold criterion by setting a non-zero value. Recommended non-default values are 0x20 for higher sensitivity with more frequent erroneous measurements, and 0x60 for reduced sensitivity and fewer erroneous measurements.

0x04

R/W	Name	Description	Initial Value
R/W	ACQ_CONFIG_REG	Acquisition mode control	0x08

Bit	Function
6	0: Enable reference process during measurement 1: Disable reference process during measurement
5	0: Use default delay for burst and free running mode 1: Use delay from MEASURE_DELAY (0x45) for burst and free running mode
4	0: Enable reference filter, averages 8 reference measurements for increased consistency 1: Disable reference filter
3	0: Enable measurement quick termination. Device will terminate distance measurement early if it anticipates that the signal peak in the correlation record will reach maximum value. 1: Disable measurement quick termination.
2	0: Use default reference acquisition count of 5. 1: Use reference acquisition count from REF_COUNT_VAL (0x12).
1:0	Mode Select Pin Function Control 00: Default PWM mode. Pull pin low to trigger measurement, device will respond with an active high output with a duration of 10us/cm. 01: Status output mode. Device will drive pin active high while busy. Can be used to interrupt host device. 10: Fixed delay PWM mode. Pulling pin low will not trigger a measurement. 11: Oscillator output mode. Nominal 31.25 kHz output. The accuracy of the silicon oscillator in the device is generally within 1% of nominal. This affects distance measurements proportionally and can be measured to apply a compensation factor.

0x1c

R/W	Name	Description	Initial Value
R/W	THRESHOLD_BYPASS	Peak detection threshold bypass	0x00

Bit	Function
7:0	0x00: Use default valid measurement detection algorithm based on the peak value, signal strength, and noise in the correlation record. 0x01-0xff: Set simple threshold for valid measurement detection. Values 0x20-0x60 generally perform well.

딜레이 자동측정, burst모드, 속도측정 모드


```

/*
 * MEASURE_DELAY -자동 측정 사이의 지연( ACQ_CONFIG_REG 셋팅 필수 )
 * (7:0) 0xc8 일때 10Hz의 반복 0x14일 때 100Hz
 *
 * default : 0x14 의 100Hz가 시간이 짧기 때문에 딜레이 지연시간이 제일 짧다.
 */

/*
 * VELOCITY - 속도 측정후 출력.
 * 현재 측정 값과 이전값의 차이를 표시합니다(2의 보수 값). 센티미터 단위로 표시합니다.
 */

```

Burst Measurements and Free Running Mode

Address	Name	Description	Initial Value
0x04	ACQ_CONFIG_REG	Acquisition mode control	0x08
0x11	OUTER_LOOP_COUNT	<u>Burst measurement count control</u>	0x00
0x45	<u>MEASURE_DELAY</u>	Delay between <u>automatic measurements</u>	0x14

The device can be configured to take multiple measurements for each measurement command or repeat indefinitely for free running mode.

OUTER_LOOP_COUNT (0x11) controls the number of times the device will retrigger itself. Values 0x00 or 0x01 result in the default one measurement per command. Values 0x02 to 0xfe directly set the repetition count. Value 0xff will enable free running mode after the host device sends an initial measurement command.

The default delay between automatic measurements corresponds to a 10 Hz repetition rate. Set bit 5 in ACQ_CONFIG_REG (0x04) to use the delay value in MEASURE_DELAY (0x45) instead. A delay value of 0x14 roughly corresponds to 100Hz.

The delay is timed from the completion of each measurement. The means that measurement duration, which varies with returned signal strength, will affect the repetition rate. At low repetition rates (high delay) this effect is small, but for lower delay values it is recommended to limit the maximum acquisition count if consistent frequency is desired.

Velocity

Address	Name	Description	Initial Value
0x09	VELOCITY	Velocity measurement output	--

The velocity measurement is the difference between the current measurement and the previous one, resulting in a signed (2's complement) 8-bit number in cm. Positive velocity is away from the device. This can be combined with free running mode for a constant measurement frequency. The default free running frequency of 10 Hz therefore results in a velocity measurement in .1 m/s.

0x45

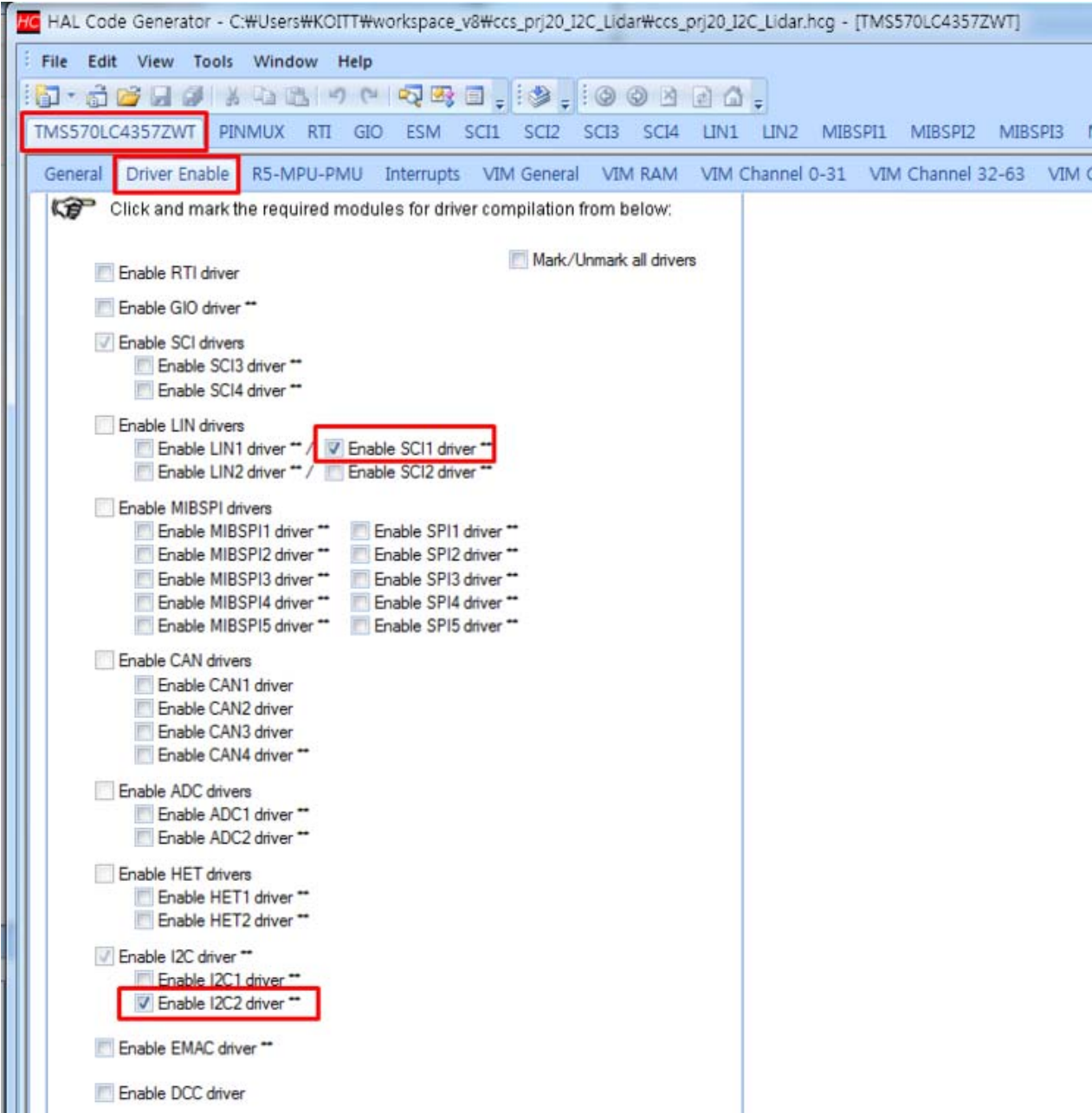
R/W	Name	Description	Initial Value
R/W	MEASURE_DELAY	Delay between automatic measurements	0x14

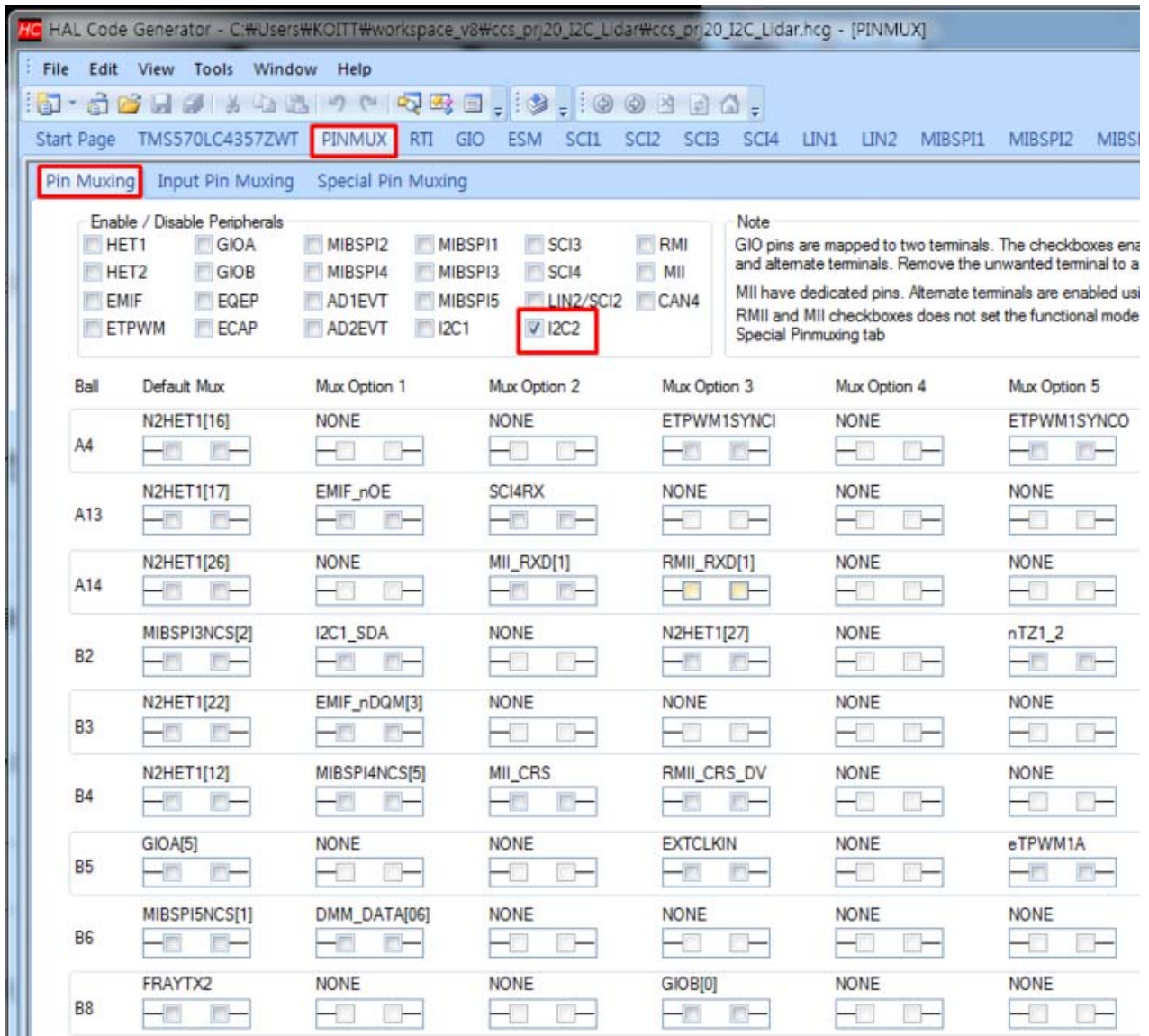
Bit	Function
7:0	Non-default delay after completion of measurement before automatic retrigger, in burst and continuous modes. ACQ_CONFIG_REG (0x04) bit 5 must be set. Value 0xc8 corresponds to 10 Hz repetition rate and 0x14 to roughly 100 Hz.

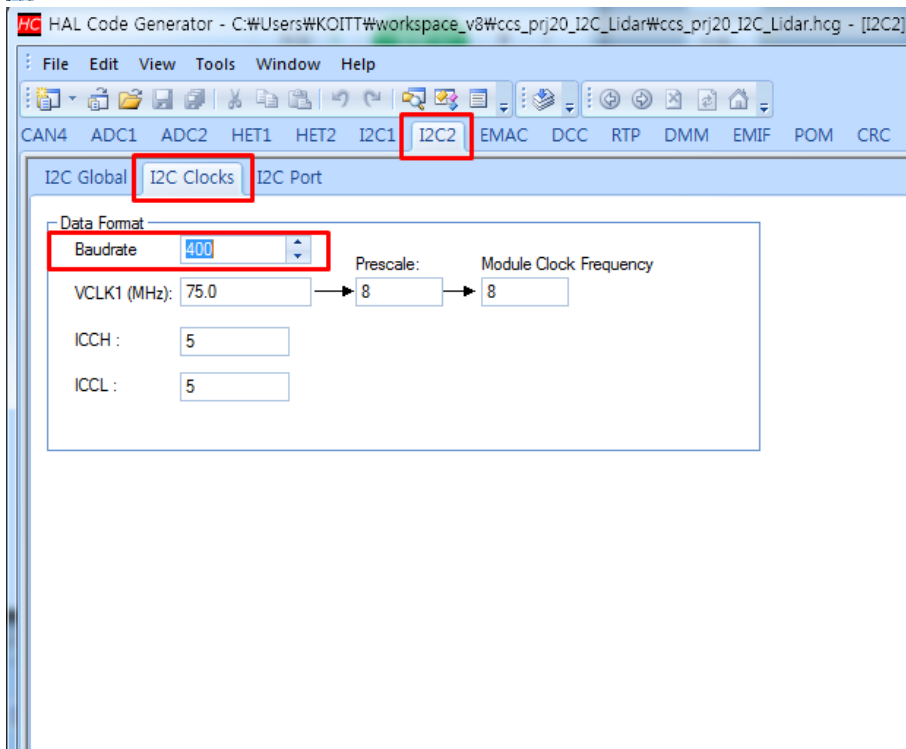
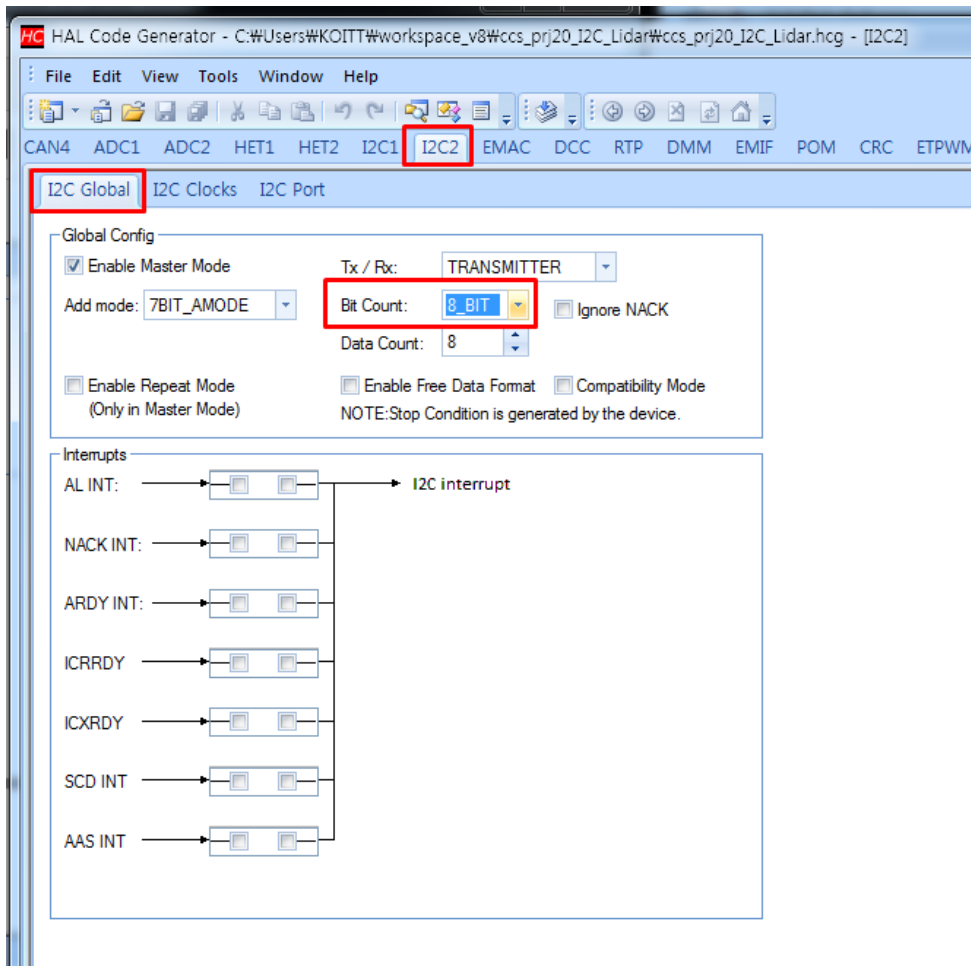
0x09

R/W	Name	Description	Initial Value
R	VELOCITY	Velocity measurement output	--

Bit	Function
7:0	Velocity measurement output. The difference between the current measurement and the previous one, signed (2's complement) value in centimeters.







<CODE_민감도 조절 완료>

```
#include <HL_hal_stdtypes.h>
#include <HL_i2c.h>
#include <HL_reg_sci.h>
#include <HL_sci.h>
#include <HL_sys_core.h>
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
```

```

#include <string.h>
#define UART    sciREG1
#define LIDAR_SLAVE_ADDR 0x62
#define ACQ_COMMAND 0x00
#define STATUS    0x01
#define SIG_COUNT_VAL 0x02
#define ACQ_CONFIG_REG 0x04
#define VELOCITY    0x09
#define PEAK_CORR    0x0c
#define FULL_DELAY_HIGH 0x0f
#define OUTER_LOOP_COUNT 0x11
#define THRESHOLD_BYPASS 0x1C
#define MEASURE_DELAY 0x45
#define PEAK_BCK 0x4c
#define READ_FROM    0x8f
// 측정 모드를 바꿔 준다. ( datasheet를 참고해서 bit를 제어해야 한다.)
/* ACQ_CONFIG_REG
 * 0x08+(1:0) 00 : Default PWM mode 10us/cm 단위로 duty
 *          01 : Status output mode :디바이스는 Busy 동안 pin active high를 구동 할 것이다.
호스트 장치를 인터럽트하는 데 사용할 수 있습니다.
10 : Fixed delay PWM mode : pin을 low로 설정하고 트리거 측정을 중지합니다.
11 : Oscillator output mode : 공칭 31.25kHz 출력.
디바이스의 실리콘 발진기의 정확도는 일반적으로 공칭의 1 % 이내입니다.
이는 거리 측정에 비례하여 영향을 미치며 보상 계수를 적용하여 측정
할 수 있습니다
0x08+(2 ) 1 : use acquisition count from REF_COUNT_VAL(0x12)
0x08+(3 ) 1 : disable measurement quick termination 0: enable measurement quick termination
0x08+(4 ) 1 : disable filter 0: enable filter(일관성을 위해 8개의 값을 참조)
0x08+(5 ) 1 : Use delay from MEASURE_DELAY(0x45)참조 0: 기본 버스트 모드 프리러닝모드 사용
0x08+(6 ) 1 : Disable process during measurement 0: enable
*/
/*
 * MEASURE_DELAY -자동 측정 사이의 지연( ACQ_CONFIG_REG 설정 필수 )
 * (7:0) 0xc8 일때 10Hz의 반복 0x14일 때 100Hz
 *
 * default : 0x14 의 100Hz가 시간이 짧기 때문에 딜레이 지연시간이 제일 짧다.
 */
uint8 bias_cnt = 0;
uint8 receives[2];
uint8 receives1[8] = { 0 };
uint16 tmp = 0;
int tmp1 = 0;
volatile int g_acc_flag;
void sciDisplayText(sciBASE_t *sci, uint8 *text, uint32 len);
void pwmSet(void);
void wait(uint32 delay);
void Lidar_without_bias(void);
void Lidar_bias(void);
void Get_Data(void);
void Lidar_enable(void);
void Lidar_verocity(void);
void wait(uint32 delay) {
    int i = 1;
    for (i = 0; i < delay; i++)
        ;
}
void sciDisplayText(sciBASE_t *sci, uint8 *text, uint32 len) {
    while (len--) {
        while ((UART->FLR & 0x4) == 4)
            ;
        sciSendByte(UART, *text++);
    }
}
void disp_set(char *str) {
    char txt_buf[256] = { 0 };

```

```

unsigned int buf_len;
sprintf(txt_buf, str);
buf_len = strlen(txt_buf);
sciDisplayText(sciREG1, (uint8 *) txt_buf, buf_len);
wait(100000);
}
void Get_Data_Ave(void) {
}
int main(void) {
uint8 txt_buf[256] = { 0 };
unsigned int buf_len;
int i;
int sum_velocity = 0;
int ave_velocity = 0;
uint32 ave = 0;
uint32 sum = 0;
sciInit();
disp_set("SCI Configuration Success!!\nWrW0");
wait(1000000);
i2cInit();
wait(1000000);
disp_set("I2C Init Success!!\nWrW0");
Lidar_enable();
disp_set("Lidar Enable Success!!\nWrW0");
wait(1000000);
for (;;) {
Get_Data();
wait(1000);
// 함수를 쉼터면 프리러닝 모드를 잘 써야하는 건지 정확히는 모르겠지만 너무 값이 들쭉날쭉 된다.
// 아마도 THRESHOLD_BYPASS의 민감도를 0x60으로 바꾸어야 할 것 같다. 굳이 쓰지 않아서 막아둔다.
//Lidar_verocity();
//wait(1000);
if (i % 50 == 0) {
ave = sum / i; //FIR필터
// ave_velocity = sum_velocity / i; //FIR필터
sciDisplayText(sciREG1, "Distance = ", 12);
ltoa(ave, (char *) txt_buf);
buf_len = strlen((const char *) txt_buf);
sciDisplayText(sciREG1, (uint8 *) txt_buf, buf_len);
// sciDisplayText(sciREG1, "WtWt", 2);
// sciDisplayText(sciREG1, "Velocity = ", 12);
// ltoa(ave_velocity, (char *) txt_buf);
// buf_len = strlen((const char *) txt_buf);
// sciDisplayText(sciREG1, (uint8 *) txt_buf, buf_len);
// sciDisplayText(sciREG1, "m/s\nWrW0", 7);
sciDisplayText(sciREG1, "WnWrW0", 4);
// sum_velocity = 0;
sum = 0;
i = 1;
}
else {
// sum_velocity += (256-tmp1); // 2의 보수의 형태라서 8bit에서 거리차를 구하면 1 0000 0000 에서 나온 보수값을 뺀다.
sum += tmp;
i++;
}
if (bias_cnt == 100) {
Lidar_bias(); //100번째에 ACQ_COMMAND에 0x04 세팅
bias_cnt = 0;
}
else {
Lidar_without_bias(); //99번은 ACQ_COMMAND에 0x03 세팅
bias_cnt++;
}
}
return 0;

```



```

}

void Lidar_enable(void) {
    uint8 tmp[5] = { 0x32, 0x08, 0x30, 0x04 };
    volatile unsigned int cnt = 7;
    disp_set("debug1WnWrW0");
    i2cSetSlaveAdd(i2cREG2, LIDAR_SLAVE_ADDR); //LIDAR_SLAVE_ADDR 0x62
    i2cSetDirection(i2cREG2, I2C_TRANSMITTER); //전송모드
    /* 마스터 모드에서만 사용가능 전송값*/
    i2cSetCount(i2cREG2, cnt + 1);
    i2cSetMode(i2cREG2, I2C_MASTER); //마스터모드
    i2cSetStop(i2cREG2); //안정성을 위해 스탑부터
    i2cSetStart(i2cREG2); //시작
    // 최대 획득 수. (주소 : 0x02 , 첫 값: 0x80)
    /*
    * 최대 획득 수가 기본 0x80이다 0:7 bit 설정 가능
    * 우리는 50 정도로 해 본다. 0x32
    */
    i2cSendByte(i2cREG2, SIG_COUNT_VAL);
    i2cSend(i2cREG2, 1, &tmp[0]); //0x80
    // 측정 모드를 바꿔 준다. ( datasheet를 참고해서 bit를 제어해야 한다.)
    // 현재 위에 모드가 어떻게 설정되어 있는지 켜다.
    // quick termination 모드가 꺼져 있다. 나머지 디폴트 모드이다.
    i2cSendByte(i2cREG2, ACQ_CONFIG_REG);
    i2cSend(i2cREG2, 1, &tmp[1]); //0x08
    wait(10000);
    // 감지의 민감도 설정 THRESHOLD_BYPASS
    /* -recommended-
    * 0x20은 민감도가 높고 에러율이 높다.
    * 0x60은 민감도가 낮고 에러율이 낮다.
    * 우리는 그러면 중간 적당히 민감도가 좋게 하여 한다.(0x25-0x30) 정도 써보자.
    */
    i2cSendByte(i2cREG2, THRESHOLD_BYPASS);
    i2cSend(i2cREG2, 1, &tmp[2]); //0x00
    wait(10000);
    // Device command
    // - Receiver Bias correction 에서 100회의 연속 측정을 시작할 때 마다 보정모드를 주기적으로 해주어야 한다라는 의미 이다.
    // (한마디로 100회가 overflow가 일어날때 0x04로 한번 바꾸고 그 후 99회는 0x03으로 돌린다.)
    // 0x00 : 디바이스를 리셋해서 레지스터의 값을 처음 기본값 으로 돌린다.
    // 0x03 : 보정모드 없이 동작을 하는 것이다. (하지만 정확도와 속도가 악영향을 받는다.)
    // 0x04 : 보정모드를 하며 동작하는 것이다.
    /* 동작에 처음이니 보정모드를 사용한다.*/
    i2cSendByte(i2cREG2, ACQ_COMMAND);
    i2cSend(i2cREG2, 1, &tmp[3]); //0x04
    wait(10000);
    /*
    * VELOCITY - 속도 측정후 출력.
    * 현재 측정 값과 기호 값 (2의 보수 값) 사이의 차이를 센티미터 단위로 표시합니다.
    */
    // i2cSendByte(i2cREG2, VELOCITY);
    // i2cSend(i2cREG2, 1, &tmp[3]); //0x04
    // wait(1000);
    disp_set("Lidar tmp 1 Enable Success!!WnWrW0");
    while (i2cIsBusBusy(i2cREG2) == true)
        ; //데이터가 오고 있는지
    while (i2cIsStopDetected(i2cREG2) == 0)
        ; //데이터가 다 왔는지
    i2cClearSCD(i2cREG2); //초기화
    wait(100000);
}

void Get_Data() {
    i2cSetSlaveAdd(i2cREG2, LIDAR_SLAVE_ADDR); //LIDAR_SLAVE_ADDR 0x62
    i2cSetDirection(i2cREG2, I2C_TRANSMITTER); //전송
    i2cSetCount(i2cREG2, 1);
    i2cSetMode(i2cREG2, I2C_MASTER); //마스터모드
    i2cSetStop(i2cREG2); //스탑부터(안정성)

```

```

i2cSetStart(i2cREG2); //시작
i2cSendByte(i2cREG2, READ_FROM); //0x8f
while (i2clsBusBusy(i2cREG2) == true)
//데이터가 오고 있는지
;
while (i2clsStopDetected(i2cREG2) == 0)
//데이터가 다 왔는지
;
i2cClearSCD(i2cREG2); //초기화
i2cSetDirection(i2cREG2, I2C_RECEIVER); //데이터 받음
i2cSetCount(i2cREG2, 2);
i2cSetMode(i2cREG2, I2C_MASTER); //마스터모드
i2cSetStart(i2cREG2); //시작
i2cReceive(i2cREG2, 2, (unsigned char *) receives);
i2cSetStop(i2cREG2); //끝
while (i2clsBusBusy(i2cREG2) == true)
//데이터가 오고 있는지
;
while (i2clsStopDetected(i2cREG2) == 0)
//데이터가 다 왔는지
;
i2cClearSCD(i2cREG2); //초기화
wait(2000);
// 받아온 값을 한한다.
tmp = receives[0] << 8; //8비트 두개를 합쳐서 2바이트로 저장
tmp |= receives[1];
g_acc_flag = 1; //다시 값을 받기 위하여
}

void Lidar_bias(void) {
volatile unsigned int cnt = 1;
unsigned char data[1] = { 0x04U };
i2cSetSlaveAdd(i2cREG2, LIDAR_SLAVE_ADDR);
i2cSetDirection(i2cREG2, I2C_TRANSMITTER);
i2cSetCount(i2cREG2, cnt + 1);
i2cSetMode(i2cREG2, I2C_MASTER);
i2cSetStop(i2cREG2);
i2cSetStart(i2cREG2);
i2cSendByte(i2cREG2, ACQ_COMMAND);
i2cSend(i2cREG2, cnt, data);
i2cSetStop(i2cREG2);
while (i2clsBusBusy(i2cREG2) == true)
;
while (i2clsStopDetected(i2cREG2) == 0)
;
i2cClearSCD(i2cREG2);
wait(30000);
}

void Lidar_without_bias(void) {
volatile unsigned int cnt = 1;
unsigned char data[1] = { 0x03U };
i2cSetSlaveAdd(i2cREG2, LIDAR_SLAVE_ADDR);
i2cSetDirection(i2cREG2, I2C_TRANSMITTER);
i2cSetCount(i2cREG2, cnt + 1);
i2cSetMode(i2cREG2, I2C_MASTER);
i2cSetStop(i2cREG2);
i2cSetStart(i2cREG2);
i2cSendByte(i2cREG2, ACQ_COMMAND);
i2cSend(i2cREG2, cnt, data);
i2cSetStop(i2cREG2);
while (i2clsBusBusy(i2cREG2) == true)
;
while (i2clsStopDetected(i2cREG2) == 0)
;
i2cClearSCD(i2cREG2);
wait(30000);
}

```

```

}

void Lidar_verocity(void) {
/*
 * VELOCITY - 속도 측정후 출력.
 * 현재 측정 값과 기호 값 (2의 보수 값) 사이의 차이를 센티미터 단위로 표시합니다.
 */
uint8 txt_buf[256] = { 0 };
unsigned int buf_len;
i2cSetSlaveAdd(i2cREG2, LIDAR_SLAVE_ADDR); //LIDAR_SLAVE_ADDR 0x62
i2cSetDirection(i2cREG2, I2C_TRANSMITTER); //전송
i2cSetCount(i2cREG2, 1);
i2cSetMode(i2cREG2, I2C_MASTER); //마스터모드
i2cSetStop(i2cREG2); //스탑부터(안정성)
i2cSetStart(i2cREG2); //시작
i2cSendByte(i2cREG2, VELOCITY); //0x09
while (i2cIsBusBusy(i2cREG2) == true)
//데이터가 오고 있는지
;
while (i2cIsStopDetected(i2cREG2) == 0)
//데이터가 다 왔는지
;
i2cClearSCD(i2cREG2); //초기화
i2cSetDirection(i2cREG2, I2C_RECEIVER); //데이터 수신
i2cSetCount(i2cREG2, 2);
i2cSetMode(i2cREG2, I2C_MASTER); //마스터모드
i2cSetStart(i2cREG2); //시작
i2cReceive(i2cREG2, 2, (unsigned char *) receives1);
i2cSetStop(i2cREG2); //끝
while (i2cIsBusBusy(i2cREG2) == true)
//데이터가 오고 있는지
;
while (i2cIsStopDetected(i2cREG2) == 0)
//데이터가 다 왔는지
;
i2cClearSCD(i2cREG2); //초기화
wait(2000);
// 받아온 값을 한한다.
tmp1 = receives1[0]; //8비트?
sprintf((char *)txt_buf, "Velrocity = %dWtWt", tmp1);
buf_len = strlen((const char*)txt_buf);
sciDisplayText(sciREG1, (uint8 *) txt_buf, buf_len);
// tmp1 = atoi((const char *) receives1); // 이렇게 받으면 안됨.
// tmp1 |= receives1[1];
g_acc_flag = 1; //다시 값을 받기 위하여
}

```

소스코드 표란에 쓰기

```

#include <HL_hal_stdtypes.h>
#include <HL_i2c.h>
#include <HL_reg_sci.h>
#include <HL_sci.h>
#include <HL_sys_core.h>
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define UART                sciREG1
#define LIDAR_SLAVE_ADDR    0x62
#define ACQ_COMMAND         0x00

```

```

#define STATUS                0x01
#define SIG_COUNT_VAL        0x02
#define ACQ_CONFIG_REG       0x04
#define VELOCITY              0x09
#define PEAK_CORR             0x0c
#define FULL_DELAY_HIGH      0x0f
#define OUTER_LOOP_COUNT     0x11
#define THRESHOLD_BYPASS      0x1C
#define MEASURE_DELAY         0x45
#define PEAK_BCK              0x4c
#define READ_FROM             0x8f

// 측정 모드를 바꿔 준다. ( datasheet를 참고해서 bit를 제어해야 한다.)
/* ACQ_CONFIG_REG
 * 0x08+(1:0) 00 : Default PWM mode 10us/cm 단위로 duty
 *
 *      01 : Status output mode :디바이스는 Busy 동안 pin active high를 구동 할 것이다.
호스트 장치를 인터럽트하는 데 사용할 수 있습니다.
10 : Fixed delay PWM mode : pin을 low로 설정하고 트리거 측정을 중지합니다.
11 : Oscillator output mode : 공칭 31.25kHz 출력.
디바이스의 실리콘 발진기의 정확도는 일반적으로 공칭의 1 % 이내입니다.
이는 거리 측정에 비례하여 영향을 미치며 보상 계수를 적용하여 측정
할 수 있습니다
0x08+(2 ) 1 : use acquisition count from REF_COUNT_VAL(0x12)
0x08+(3 ) 1 : disable measurement quick termination 0: enable measurement quick termination
0x08+(4 ) 1 : disable filter 0: enable filter(일관성을 위해 8개의 값을 참조)
0x08+(5 ) 1 : Use delay from MEASURE_DELAY(0x45)참조 0: 기본 버스트 모드 프리러닝모드 사용
0x08+(6 ) 1 : Disable process during measurement 0: enable
 */
/*
 * MEASURE_DELAY -자동 측정 사이의 지연( ACQ_CONFIG_REG 셋팅 필수 )
 * (7:0) 0xc8 일때 10Hz의 반복 0x14일 때 100Hz
 *
 * default : 0x14 의 100Hz가 시간이 짧기 때문에 딜레이 지연시간이 제일 짧다.
 */

uint8 bias_cnt = 0;
uint8 receives[2];
uint8 receives1[8] = { 0 };
uint16 tmp = 0;
int tmp1 = 0;

volatile int g_acc_flag;

void sciDisplayText(sciBASE_t *sci, uint8 *text, uint32 len);
void pwmSet(void);
void wait(uint32 delay);
void Lidar_without_bias(void);
void Lidar_bias(void);
void Get_Data(void);
void Lidar_enable(void);
void Lidar_verocity(void);

void wait(uint32 delay) {
    int i = 1;
    for (i = 0; i < delay; i++)

```

```

        ;
    }

void sciDisplayText(sciBASE_t *sci, uint8 *text, uint32 len) {
    while (len--) {
        while ((UART->FLR & 0x4) == 4)
            ;
        sciSendByte(UART, *text++);
    }
}

void disp_set(char *str) {
    char txt_buf[256] = { 0 };
    unsigned int buf_len;
    sprintf(txt_buf, str);
    buf_len = strlen(txt_buf);
    sciDisplayText(sciREG1, (uint8 *) txt_buf, buf_len);
    wait(1000000);
}

void Get_Data_Ave(void) {

}

int main(void) {
    uint8 txt_buf[256] = { 0 };
    unsigned int buf_len;
    int i;
    int sum_velocity = 0;
    int ave_velocity = 0;
    uint32 ave = 0;
    uint32 sum = 0;

    sciInit();
    disp_set("SCI Configuration Success!!\n\r\0");
    wait(100000000);

    i2cInit();
    wait(100000000);
    disp_set("I2C Init Success!!\n\r\0");

    Lidar_enable();
    disp_set("Lidar Enable Success!!\n\r\0");
    wait(10000000);

    for (;;) {
        Get_Data();
        wait(1000);
        // 함수를 실행하면 프리러닝 모드를 잘 써야하는 건지 정확히는 모르겠지만 너무 값이 들쭉날쭉 된다.
        // 아마도 THRESHOLD_BYPASS의 민감도를 0x60으로 바꾸어야 할 것 같다. 굳이 쓰지 않아서 막아둔다.
        //Lidar_velocity();
        //wait(1000);

        if (i % 50 == 0) {
            ave = sum / i; //FIR필터
            //
            ave_velocity = sum_velocity / i; //FIR필터

            sciDisplayText(sciREG1, "Distance = ", 12);

```

```

        ltoa(ave, (char *) txt_buf);
        buf_len = strlen((const char *) txt_buf);
        sciDisplayText(sciREG1, (uint8 *) txt_buf, buf_len);

//        sciDisplayText(sciREG1, "\t\t", 2);
//        sciDisplayText(sciREG1, "Velocity = ", 12);
//        ltoa(ave_velocity, (char *) txt_buf);
//        buf_len = strlen((const char *) txt_buf);
//        sciDisplayText(sciREG1, (uint8 *) txt_buf, buf_len);
//        sciDisplayText(sciREG1, "m/s\n\r\0", 7);

        sciDisplayText(sciREG1, "\n\r\0", 4);

//        sum_velocity = 0;
//        sum = 0;
//        i = 1;
//    }
//    else {
//        sum_velocity += (256-tmp1); // 2의 보수의 형태라서 8bit에서 거리차를 구하면 1 0000 0000 에서 나온 보수값을 뺀다.
//        sum += tmp;
//        i++;
//    }

    if (bias_cnt == 100) {
        Lidar_bias(); //100번째에 ACQ_COMMAND에 0x04 세팅
        bias_cnt = 0;
    }
    else {
        Lidar_without_bias(); //99번은 ACQ_COMMAND에 0x03 세팅
        bias_cnt++;
    }
}
return 0;
}

```

```

void Lidar_enable(void) {

    uint8 tmp[5] = { 0x32, 0x08, 0x30, 0x04 };
    volatile unsigned int cnt = 7;

    disp_set("debug1\n\r\0");

    i2cSetSlaveAdd(i2cREG2, LIDAR_SLAVE_ADDR); //LIDAR_SLAVE_ADDR 0x62
    i2cSetDirection(i2cREG2, I2C_TRANSMITTER); //전송모드
    /* 마스터 모드에서만 사용가능 전송값*/
    i2cSetCount(i2cREG2, cnt + 1);
    i2cSetMode(i2cREG2, I2C_MASTER); //마스터모드
    i2cSetStop(i2cREG2); //안정성을 위해 스탑부터
    i2cSetStart(i2cREG2); //시작

    // 최대 획득 수. (주소 : 0x02 , 첫 값: 0x80)
    /*
    * 최대 획득 수가 기본 0x80이다 0:7 bit 설정 가능
    */
}

```



```

* 우리는 50 정도로 해 본다. 0x32

*/

i2cSendByte(i2cREG2, SIG_COUNT_VAL);

i2cSend(i2cREG2, 1, &tmp[0]); //0x80

// 측정 모드를 바꿔 준다. ( datasheet를 참고해서 bit를 제어해야 한다.)
// 현재 위에 모드가 어떻게 설정되어 있는지 켜다.
// quick termination 모드가 꺼져 있다. 나머지 디폴트 모드이다.

i2cSendByte(i2cREG2, ACQ_CONFIG_REG);

i2cSend(i2cREG2, 1, &tmp[1]); //0x08

wait(10000);

// 감지의 민감도 설정 THRESHOLD_BYPASS
/* -recommended-
* 0x20은 민감도가 높고 에러율이 높다.
* 0x60은 민감도가 낮고 에러율이 낮다.
* 우리는 그러면 중간 적당히 민감도가 좋게 하여 한다.(0x25-0x30) 정도 써보자.
*/

i2cSendByte(i2cREG2, THRESHOLD_BYPASS);

i2cSend(i2cREG2, 1, &tmp[2]); //0x00

wait(10000);

// Device command
// - Receiver Bias correction 에서 100회의 연속 측정을 시작할 때 마다 보정모드를 주기적으로 해주어야 한다라는 의미 이다.
// (한마디로 100회가 overflow가 일어날때 0x04로 한번 바꾸고 그 후 99회는 0x03으로 돌린다.)

// 0x00 : 디바이스를 리셋해서 레지스터의 값을 처음 기본값 으로 돌린다.
// 0x03 : 보정모드 없이 동작을 하는 것이다. (하지만 정확도와 속도가 악영향을 받는다.)
// 0x04 : 보정모드를 하며 동작하는 것이다.
/* 동작에 처음이니 보정모드를 사용한다.*/

i2cSendByte(i2cREG2, ACQ_COMMAND);

i2cSend(i2cREG2, 1, &tmp[3]); //0x04

wait(10000);

/*
* VELOCITY - 속도 측정후 출력.
* 현재 측정 값과 기호 값 (2의 보수 값) 사이의 차이를 센티미터 단위로 표시합니다.
*/

// i2cSendByte(i2cREG2,VELOCITY);
// i2cSend(i2cREG2, 1, &tmp[3]); //0x04
// wait(1000);

disp_set("Lidar tmp 1 Enable Success!!\n\r\0");

while (i2cIsBusBusy(i2cREG2) == true)
    ; //데이터가 오고 있는지

while (i2cIsStopDetected(i2cREG2) == 0)
    ; //데이터가 다 왔는지

i2cClearSCD(i2cREG2); //초기화

wait(100000);

}

void Get_Data() {

    i2cSetSlaveAdd(i2cREG2, LIDAR_SLAVE_ADDR); //LIDAR_SLAVE_ADDR 0x62

    i2cSetDirection(i2cREG2, I2C_TRANSMITTER); //전송

    i2cSetCount(i2cREG2, 1);

    i2cSetMode(i2cREG2, I2C_MASTER); //마스터모드

    i2cSetStop(i2cREG2); //스탑부터(안정성)

```

```

    i2cSetStart(i2cREG2); //시작
    i2cSendByte(i2cREG2, READ_FROM); //0x8f
    while (i2cIsBusBusy(i2cREG2) == true)
//데이터가 오고 있는지
        ;
    while (i2cIsStopDetected(i2cREG2) == 0)
//데이터가 다 왔는지
        ;
    i2cClearSCD(i2cREG2); //초기화
    i2cSetDirection(i2cREG2, I2C_RECEIVER); //데이터 받음
    i2cSetCount(i2cREG2, 2);
    i2cSetMode(i2cREG2, I2C_MASTER); //마스터모드
    i2cSetStart(i2cREG2); //시작
    i2cReceive(i2cREG2, 2, (unsigned char *) receives);
    i2cSetStop(i2cREG2); //끝
    while (i2cIsBusBusy(i2cREG2) == true)
//데이터가 오고 있는지
        ;
    while (i2cIsStopDetected(i2cREG2) == 0)
//데이터가 다 왔는지
        ;
    i2cClearSCD(i2cREG2); //초기화
    wait(2000);
    // 받아온 값을 한한다.
    tmp = receives[0] << 8; //8비트 두개를 합쳐서 2바이트로 저장
    tmp |= receives[1];

    g_acc_flag = 1; //다시 값을 받기 위하여
}

void Lidar_bias(void) {
    volatile unsigned int cnt = 1;
    unsigned char data[1] = { 0x04U };
    i2cSetSlaveAdd(i2cREG2, LIDAR_SLAVE_ADDR);
    i2cSetDirection(i2cREG2, I2C_TRANSMITTER);
    i2cSetCount(i2cREG2, cnt + 1);
    i2cSetMode(i2cREG2, I2C_MASTER);
    i2cSetStop(i2cREG2);
    i2cSetStart(i2cREG2);
    i2cSendByte(i2cREG2, ACQ_COMMAND);
    i2cSend(i2cREG2, cnt, data);
    i2cSetStop(i2cREG2);
    while (i2cIsBusBusy(i2cREG2) == true)
        ;
    while (i2cIsStopDetected(i2cREG2) == 0)
        ;
    i2cClearSCD(i2cREG2);
    wait(30000);
}

void Lidar_without_bias(void) {
    volatile unsigned int cnt = 1;
    unsigned char data[1] = { 0x03U };
    i2cSetSlaveAdd(i2cREG2, LIDAR_SLAVE_ADDR);
    i2cSetDirection(i2cREG2, I2C_TRANSMITTER);
    i2cSetCount(i2cREG2, cnt + 1);
    i2cSetMode(i2cREG2, I2C_MASTER);

```

```

    i2cSetStop(i2cREG2);
    i2cSetStart(i2cREG2);
    i2cSendByte(i2cREG2, ACQ_COMMAND);
    i2cSend(i2cREG2, cnt, data);
    i2cSetStop(i2cREG2);
    while (i2cIsBusBusy(i2cREG2) == true)
        ;
    while (i2cIsStopDetected(i2cREG2) == 0)
        ;
    i2cClearSCD(i2cREG2);
    wait(30000);
}

void Lidar_verocity(void) {
    /*
     * VELOCITY - 속도 측정후 출력.
     * 현재 측정 값과 기호 값 (2의 보수 값) 사이의 차이를 센티미터 단위로 표시합니다.
     */
    uint8 txt_buf[256] = { 0 };
    unsigned int buf_len;

    i2cSetSlaveAdd(i2cREG2, LIDAR_SLAVE_ADDR); //LIDAR_SLAVE_ADDR 0x62
    i2cSetDirection(i2cREG2, I2C_TRANSMITTER); //전송
    i2cSetCount(i2cREG2, 1);
    i2cSetMode(i2cREG2, I2C_MASTER); //마스터모드
    i2cSetStop(i2cREG2); //스탑부터(안정성)
    i2cSetStart(i2cREG2); //시작
    i2cSendByte(i2cREG2, VELOCITY); //0x09
    while (i2cIsBusBusy(i2cREG2) == true)
//데이터가 오고 있는지
        ;
    while (i2cIsStopDetected(i2cREG2) == 0)
//데이터가 다 왔는지
        ;
    i2cClearSCD(i2cREG2); //초기화
    i2cSetDirection(i2cREG2, I2C_RECEIVER); //데이터 수신
    i2cSetCount(i2cREG2, 2);
    i2cSetMode(i2cREG2, I2C_MASTER); //마스터모드
    i2cSetStart(i2cREG2); //시작
    i2cReceive(i2cREG2, 2, (unsigned char *) receives1);
    i2cSetStop(i2cREG2); //끝
    while (i2cIsBusBusy(i2cREG2) == true)
//데이터가 오고 있는지
        ;
    while (i2cIsStopDetected(i2cREG2) == 0)
//데이터가 다 왔는지
        ;
    i2cClearSCD(i2cREG2); //초기화
    wait(2000);
    // 받아온 값을 한한다.
    tmp1 = receives1[0]; //8비트?
    sprintf((char *)txt_buf, "Velocity = %d\t\t", tmp1);
    buf_len = strlen((const char*)txt_buf);
    sciDisplayText(sciREG1, (uint8 *) txt_buf, buf_len);
}

```

```
// tmp1 = atoi((const char *) receives1); // 이렇게 받으면 안됨.  
// tmp1 |= receives1[1];  
    g_acc_flag = 1; //다시 값을 받기 위하여  
}
```
