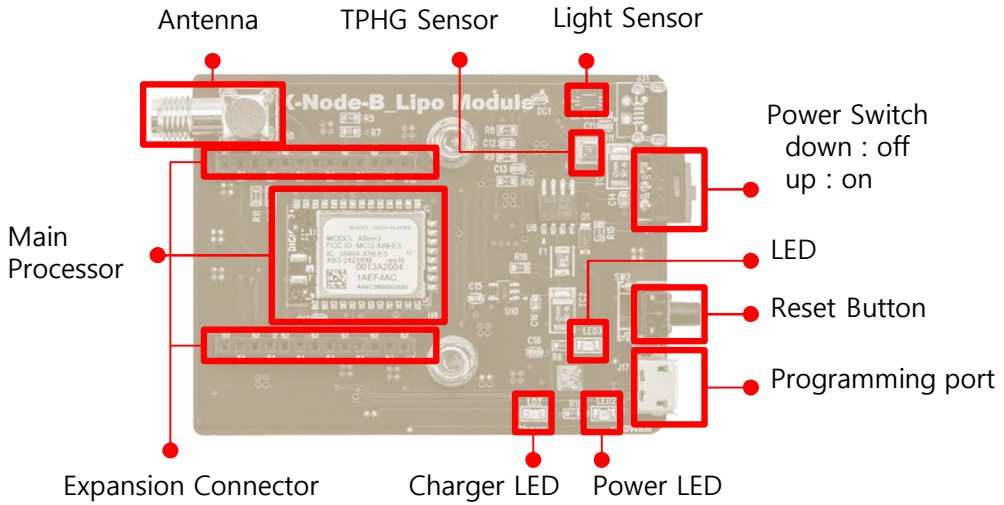
XNode로 배우는

저전력 무선 네트워크 프로그래밍

5 기본 센서 제어

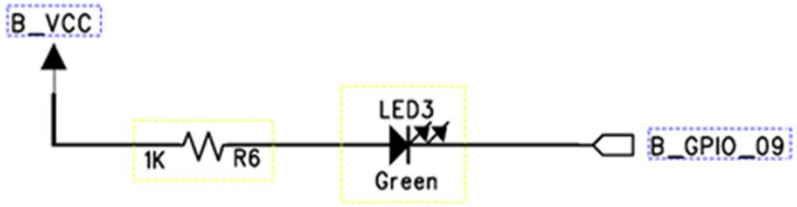
하드웨어 구성 – B Type

- Xnode B type에는 다음 센서가 기본 내장됨
  - Led
    - 상태 알림용 Led
  - 배터리 전압 센서
    - 배터리 잔량 모니터링
  - 빛 센서
    - 주변 빛의 밝기 모니터링
  - 환경 센서
    - 공기 중 섭씨온도 및 상대 습도, 기압, 가스(에탄올, 유기화합물) 모니터링



LED – B Type

- LED
  - 한쪽은 VCC, 반대쪽은 프로세서의 GPIO 9에 연결됨



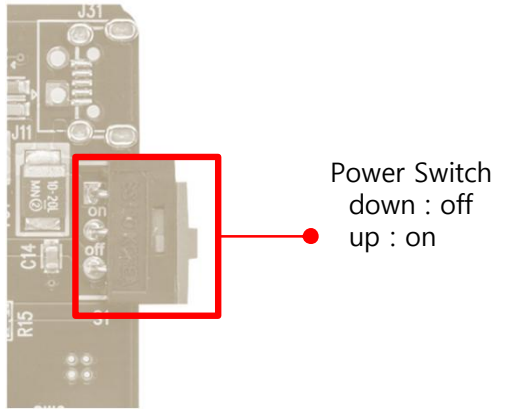
  - LOW를 출력하면 LED가 켜지고 HIGH를 출력하면 LED가 꺼짐

XNode	device
GPIO 9	LED

GPIO 9 output	LED Action
HIGH	OFF
LOW	ON

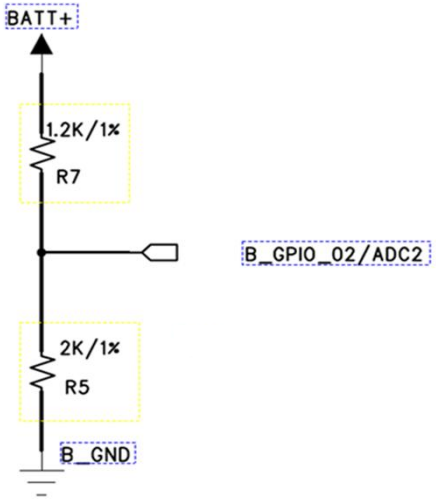
배터리 잔량 – B Type

- XNode 전원 공급
  - XNode는 배터리를 사용하여 전원 공급
  - 배터리는 USB 포트 기반으로 충전
  - 전원 스위치를 On 하여 사용
  - 배터리는 공칭전압 3.7V인 리튬폴리머 배터리를 사용



배터리 잔량 – B Type

- ADC로 배터리 전압을 측정해 현재 배터리 상태 인식
  - 배터리 전원 흐름 사이에 분배 저항을 구성한 후 프로세서의 ADC 2채널에 연결됨
    - Vout = (R2 / (R1 + R2)) * Vin
  - R1 = 1.2K, R2 = 2K
    - Vin = 3.2V
      - Vout = (2 / (1.2 + 2)) * 3.2 = 2V



배터리 잔량 – B Type

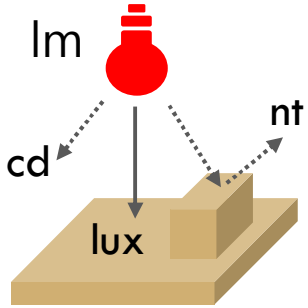
- 배터리 전압은 전압 분배 회로에 의해 전압 강하된 상태로 인식
  - ADC는 측정 전압에 대한 디지털 변환 레벨을 반환하므로 ADC 참조 전압과 해상도를 이용해 전압으로 변환
    - XNode ADC의 참조 전압은 3.3V이고 해상도는 12bit (0~4095)
    - 변환식 : volt = (adc*3.3/4095)*((1.2+2)/2)
  - 배터리 전압에 따라 최대 측정값은 4.2V이지만, 분배 저항과 배터리에 따라 오차가 있음
    - 최소 측정값이 3.2V 이하면 XNode 동작을 보장할 수 없으므로 배터리를 충전할 것

XNode	device
GPIO 2 / ADC2	Battery

Max ADC Voltage	4.2V
Min ADC Voltage	3.2V

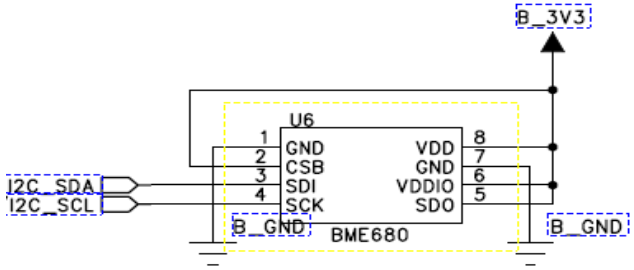
빛 센서

- 빛의 밝기 관련 단위
  - 광속: 광원에 의해 초당 방 방출되는 빛의 총량
    - 인간의 눈이 파장에 따라 서로 다르게 감응하므로 단위로는 와트(Watt) 대신 루멘(lm) 사용
  - 조도: 대상 면에 도달하는 빛의 양
    - 비치는 면적과 광속의 비율로 결정되며 단위는 럭스(lux)
  - 광도: 광원에서 어느 방향으로의 빛의 세기
    - 단위면적을 일정 시간 통과는 광속의 크기에 따라 결정하며 단위 칸델라(cd)
  - 휘도: 빛이 반사는 반사면의 밝기 (눈부심 정도)
    - 단위는 니트(nt)



빛 센서

- 빛 센서
  - 빛 센서인 BH1750는 프로세서의 I2C 핀에 연결됨

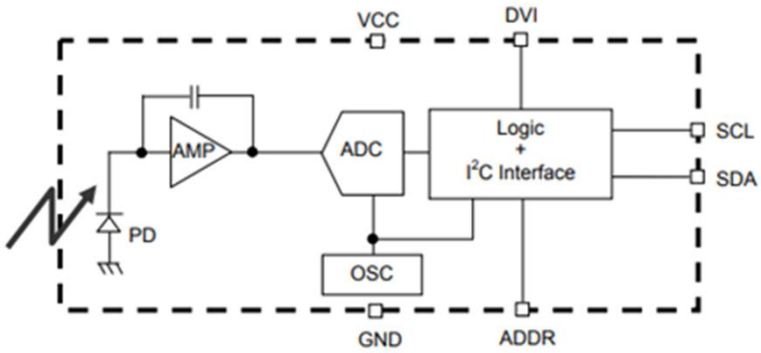


  - 주소가 0x23인 빛 센서는 센서에 도달한 빛의 양을 럭스(lux) 값으로 출력

Device	I2C Address	Range
Light 센서	0x23	1 ~ 65535 lx

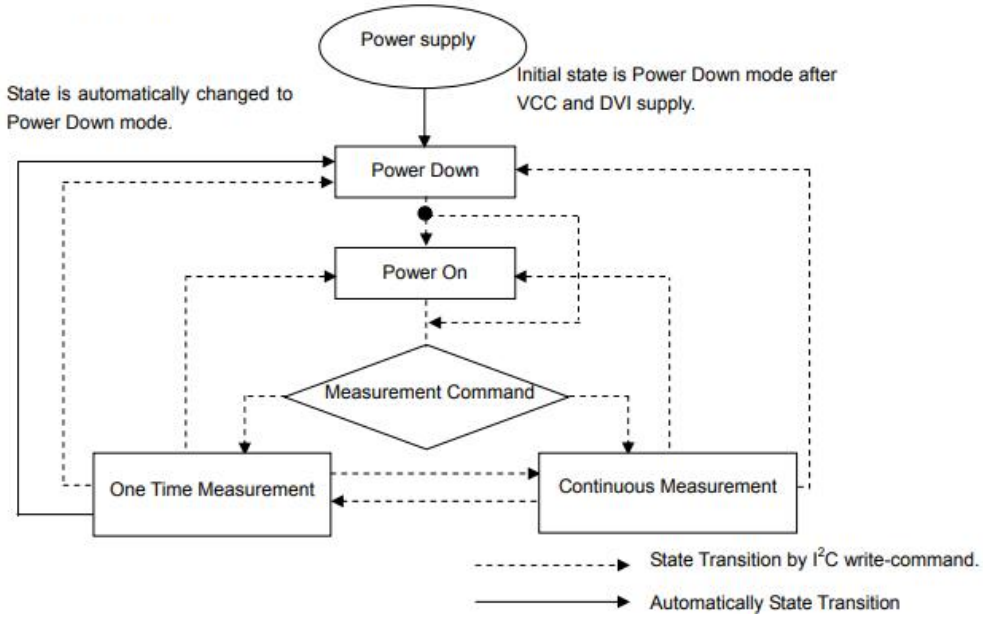
빛 센서

- 16비트 ADC가 내장되어 있어 아날로그를 디지털로 변환하는 복잡한 계산 불필요
  - 빛의 밝기를 포토다이오드가 아날로그 전압으로 출력하면 증폭기로 증폭한 후 ADC에 전달
  - ADC는 아날로그 전압에 대응하는 디지털 값을 계산한 후 로직에 전달
  - 로직은 디지털 값으로 lux 계산을 수행한 후 I2C로 출력



빛 센서

- 일회성 측정과 연속 측정
  - 일회성 측정은 측정 후 자동으로 전원이 꺼지고 연속 측정은 제어 명령으로 전원 제어



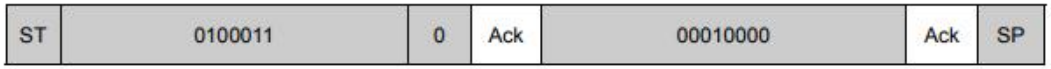
빛 센서

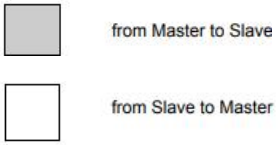
- 제어 명령

구분	코드	설명
Power Down	0x00	측정 중단
Power On	0x01	측정을 위한 대기
Reset	0x07	데이터 레지스터 초기화
Continuously H-Resolution Mode	0x10	120ms 마다 1lx 해상도로 측정 시작
Continuously H-Resolution Mode2	0x11	120ms 마다 0.5lx 해상도로 측정 시작
Continuously L-Resolution Mode	0x13	16ms 마다 4lx 해상도로 측정 시작
One Time H-Resolution Mode	0x20	120ms 동안 1lx 해상도로 측정 후 종료
One Time H-Resolution Mode2	0x21	120ms 동안 0.5lx 해상도로 측정 후 종료
One Time L-Resolution Mode	0x23	16ms 동안 4lx 해상도로 측정 후 종료

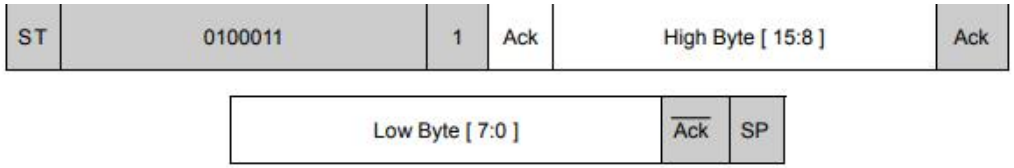
빛 센서

- 측정 절차 (H-Mode 연속 측정 예)
  - 센서에 H-Mode 연속 측정 (0x10) 명령 전달





  - 1차 측정이 완료될 때까지 대기 (최대 180ms)
  - 측정 결과 읽기
    - 상위 바이트, 하위 바이트 순으로 2바이트 반환

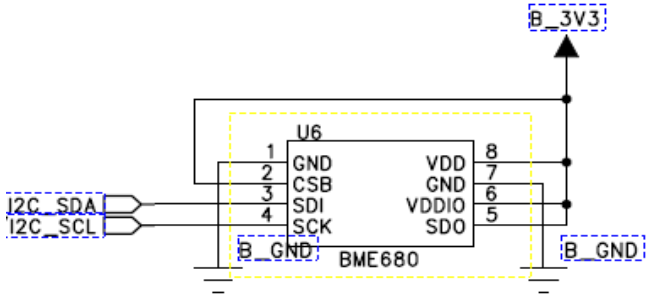


빛 센서

- 결과 변환
  - H-Mode
    - (high << 8 | low) / 1.2
  - H-Mode2
    - (high << 8 | low) / (1.2 * 2)

환경 센서

- 환경 센서
  - 환경 센서인 BME680는 I2C를 통해 프로세서에 연결됨



  - 주소가 0x77인 환경 센서는 온도, 습도 외에 기압, 공기 질(가스) 측정값을 디지털로 전달
    - 칩에 포함된 보정 값 및 제조업체에서 제공하는 공식을 통해 결과 계산

Device	I2C Address	Temperature	Humidity	Pressure	Gas
TPHG 센서	0x77	-40~85 °C	0~100% r.H	300~1100 hPa	50~50000up ohm

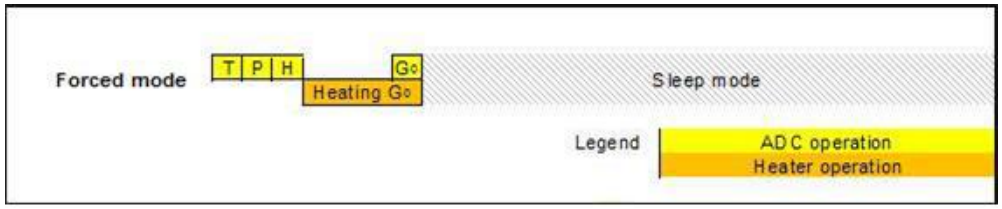
환경 센서

- 저전력 지원을 위해 포스(forced) 모드와 슬립(sleep) 모드로 나눠 운영
  - 센서에 전원이 공급되면 절전모드에서 시작
  - 측정 동안 요청한 변경 요구는 지연되거나 무시됨
  - 최초 모든 제어 레지스터의 초기화 필요

수행 모드	특징
포스 모드	- 단일 온도, 습도, 기압, 가스 측정 - 측정이 끝나면 슬립 모드로 전환 - 내장된 가스 센서 히터는 가스 측정을 할 때만 동작
슬립 모드	- 완료된 측정에 대한 ADC 수행 - 최소의 전력량 소비

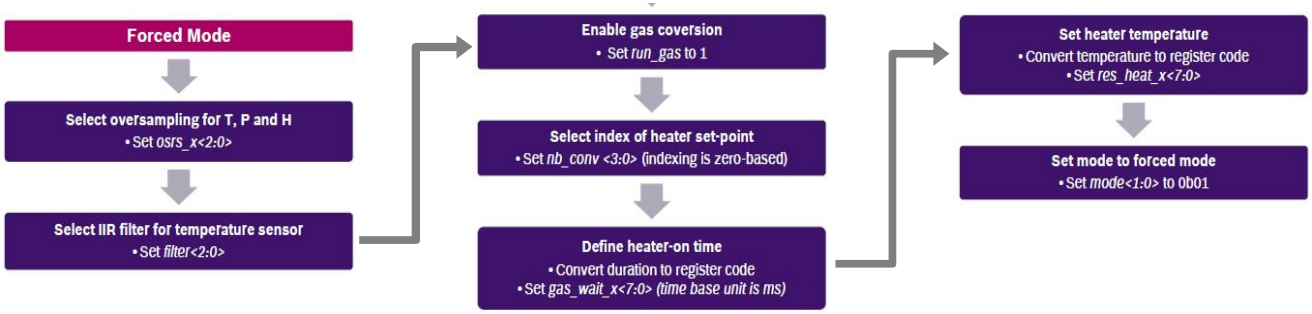
환경 센서

- 측정 개요
  - 초기화가 완료되면 포스 모드에서 순차적으로 측정
    - 온도(Temperature), 기압(Prssure), 습도(Humidity), 가스(Gas conversion) 순
  - 가스 센서 히터는 최대 10개의 온도 설정 및 가열 지속 시간을 해당 레지스터(G0 ~ G9)에 저장 가능
  - 온도, 기압, 습도 측정 후 가스 센서 히터 가열 후 가스 측정
  - 슬립 모드에서 ADC 처리



환경 센서

- 측정 절차
  - OSRS_x 레지스터를 통해 온도, 기압, 습도 측정에 필요한 최대 샘플링 횟수 선택
  - filter 레지스터를 통해 짧은 시간 발생한 급격한 변화를 제거할 IIR 필더 선택
  - run_gas 레지스터를 통해 가스 측정 여부 선택
  - 가스를 측정한다면 nb_conv 레지스터를 통해 히터 정보를 저장할 레지스터 선택
    - gas_wait_x 레지스터를 통해 밀리초 단위 히터 유지 시간 설정
    - res_heat_x 레지스터를 통해 히터 온도 설정
  - mode 레지스터를 통해 포스 모드 설정



환경 센서

- IIR 필더
  - IIR 필터 출력 해상도는 20비트로 마지막 측정값 유지됨
  - 온도 및 압력 데이터에는 적용되지만, 습도 및 가스 데이터에는 적용되지 않음
    - 온도 및 압력 결과 레지스터는 측정이 끝날 때 동시에 업데이트됨
  - ADC 데이터는 필터링 된 다음 데이터 레지스터로 로드
- 가스히터 전류 제어
  - 요구된 히터 온도를 얻기 위해 히터 저항에 충분한 전류를 공급할 히터 제어 블록 포함
  - 히터 저항값을 주기적으로 측정하고 DAC에서 공급된 mA 단위 전륫값을 조정하는 제어 루프가 있음
  - 목표 히터 온도에 대한 초기 히터 전류를 설정하여 속도를 높일 수 있음
    - 제어 루프가 몇 번의 반복 후에 전류를 찾으므로 이 단계는 선택 사항

환경 센서

- 초기화 및 운영을 위한 BME680 주요 Register
  - 모드 선택

Register	Address	Content<bit position>	Description
ctrl_meas	0x74	mode<1:0>	00 : sleep mode 01 : forced mode

  - 초기화를 위해 센서 리셋 (이후 0.01초 대기)

Register	Address	Content<bit position>	Description
reset	0xE0	reset<7:0>	0xB6 : reset 0x00 : Default

환경 센서

- 습도 센서 최대 샘플링 수 설정

Register	Address	Content<bit position>	Description
ctrl_humi	0x72	osrs_h<2:0>	000 : skip 001 : oversampling x1 010 : oversampling x2 011 : oversampling x4 100 : oversampling x5 101 , others : oversampling x 16

- 온도 센서 최대 샘플링 수 설정

Register	Address	Content<bit position>	Description
ctrl_meas	0x74	osrs_t<7:5>	000 : skip 001 : oversampling x1 010 : oversampling x2 011 : oversampling x4 100 : oversampling x5 101 , others : oversampling x 16

환경 센서

- 기압 센서 최대 샘프링 수 설정

Register	Address	Content<bit position>	Description
ctrl_meas	0x74	osrs_p<4:2>	000 : skip 001 : oversampling x1 010 : oversampling x2 011 : oversampling x4 100 : oversampling x8 101 , others : oversampling x 16

- IIR 필터 설정

Register	Address	Content<bit position>	Description (Filter coefficient)
config	0x75	filter<4:2>	000 : 0 001 : 1 010 : 3 011 : 7 100 : 15 101 : 36 110 : 63 111 : 127

환경 센서

- 가스 히터 목표 저항 설정

Heater set-point	Register	Address	Content	Description
0~9	res_heat_x x is from 0 to 9	0x5A~0x63	res_heat_x<7:0> x is from 0 to 9	목표 히터 저항을 얻기 위해 저장해야 하는 10 진수 값

- 가스 측정을 위해 히터가 데워질 때까지의 대기 시간 설정

Heater set-point	Register	Address	Content	Description
0~9	gas_wait_x x is from 0 to 9	0x64~0x6D	gas_wait_x<5:0> x is from 0 to 9	1ms 단계 크기의 64 타이머 값, 모두 0은 대기 없음 의미
0~9	gas_wait_x x is from 0 to 9	0x64~0x6D	gas_wait_x<7:6> x is from 0 to 9	가스 센서 대기시간 배수 계수 00 : 1 01 : 4 10 : 16 11 : 64

환경 센서

- 사용할 히터 설정 선택

Register	Address	Content	Description
ctrl_gas_1	0x71	nb_conv<3:0>	0000 : 0 0001 : 1 0010 : 2 0011 : 3 0100 : 4 0101 : 5 0110 : 6 0111 : 7 1000 : 8 1001 : 9

- 가스 측정 시작 설정

Register	Address	Content	Description
ctrl_gas_1	0x71	run_gas<4>	1일 때 가스 측정 시작

환경 센서

- ADC 측정 상태

Register	Address	Content	Description
meas_status_0	0x1D	new_data_0<7:0>	최상위 비트가 1이면 측정 완료

- 저수준 기압 데이터

Register	Address	Content<bit position>	Description
press_msb	0x1F	press_msb<7:0>	기압 측정 출력 데이터의 MSB 부분 [19:12]
press_lsb	0x20	press_lsb<7:0>	기압 측정 출력 데이터의 LSB 부분 [11:4]
press_xlsb	0x21	press_xlsb<7:4>	기압 측정 출력 데이터의 하위 LSB 부분 [3:0]

- 저수준 온도 데이터

Register	Address	Content<bit position>	Description
temp_msb	0x22	temp_msb<7:0>	온도 측정 출력 데이터의 MSB 부분 [19:12]
temp_lsb	0x23	temp_lsb<7:0>	온도 측정 출력 데이터의 LSB 부분 [11:4]
temp_xlsb	0x24	temp_lsb<7:4>	온도 측정 출력 데이터의 하위 LSB 부분 [3:0]

환경 센서

- 저수준 습도 데이터

Register	Address	Content<bit position>	Description
hum_msb	0x25	hum_msb<7:0>	습도 측정 출력 데이터의 MSB 부분 [15:8]
hum_lsb	0x26	hum_lsb<7:0>	습도 측정 출력 데이터의 LSB 부분 [7:0]

- 저수준 가스 데이터

Register	Address	Content<bit position>	Description
gas_r_msb	0x2A	gas_r<7:0>	가스 저항 측정 출력 데이터의 MSB 부분 [9:2]
gas_r_lsb	0x2B	gas_r<7:6>	가스 저항 측정 출력 데이터의 LSB 부분 [1:0]

Register	Address	Content	Description
gas_r_lsb	0x2B	gas_valid_r<5>	실제 가스 측정 완료 비트

Register	Address	Content	Description
gas_r_lsb	0x2B	heat_stab_r<4>	목표 히터 저항에 대한 히터 온도 안정성 비트

환경 센서

- 저수준 데이터 보정 계수

Register	Address	Content<byte>	Description
coeff_addr1	0x89	25	보정 계수의 하위 25바이트
coeff_addr2	0xE1	16	보정 계수의 상위 16바이트

- 41바이트[F0:E1, A1:89] 중 인덱스 기준 1(8A) ~ 38(EE)까지 사용
- 인덱스 26(E2)은 H2L과 H1L에 공통으로 사용하며 H1L은 [7:4]만 사용

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
89	8A	8B	8C	8D	8E	8F	90	91	92	93	94	95	96	97	98	99	9A	9B	9C	9D	9E	9F	A0	A1	E1	E2	E3	E4	E5	E6	E7	E8	E9	EA	EB	EC	ED	EE	EF	F0
	T2	T2	T3		P1	P1	P2	P2	P3		P4	P4	P5	P5	P7	P6			P8	P8	P9	P9	P10		H2	H2	H1	H3	H4	H5	H6	H7	T1	T1	G2	G2	G1	G3		
	L	M			L	M	L	M			L	M	L	M					L	M	L	M			M	L	M						L	M	L	M				
																										H1														

환경 센서

- 보정 계수와 측정 결과로부터 섭씨온도 계산
  - var1 = ((temp_adc / 16384.0) - (par_t1 / 1024.0)) * par_t2
  - var2 = (((temp_adc / 131072.0) - (par_t1 / 8192.0)) * ((temp_adc / 131072.0) - (par_t1 / 8192.0))) * (par_t3 * 16.0)
  - t_fine = var1 + var2
  - temp_comp = t_fine / 5120.0
  - par_t1 ~ par_t3: 보정 계수, temp_adc: 저수준 측정 데이터, temp_comp: 섭씨온도

Variable name	Register address (LSB/MSB)
par_t1	0xE9 / 0xEA
par_t2	0x8A / 0x8B
par_t3	0x8C
temp_adc	0x24<7:4> / 0x23 / 0x22

환경 센서

- 보정 계수와 측정 결과로부터 기압(hPs) 계산
  - var1 = (t_fine / 2.0) - 64000.0
  - var2 = var1 * var1 * (par_p6 / 131072.0)
  - var2 = var2 + (var1 * par_p5 * 2.0)
  - var2 = (var2 / 4.0) + (par_p4 * 65536.0)
  - var1 = (((par_p3 * var1 * var1) / 16384.0) + (par_p2 * var1)) / 524288.0
  - var1 = (1.0 + (var1 / 32768.0)) * par_p1
  - press_comp = 1048576.0 - press_adc
  - press_comp = ((press_comp - (var2 / 4096.0)) * 6250.0) / var1
  - var1 = (par_p9 * press_comp * press_comp) / 2147483648.0
  - var2 = press_comp * (par_p8 / 32768.0)
  - var3 = (press_comp / 256.0) * (press_comp / 256.0) * (press_comp / 256.0) * (par_p10 / 131072.0)
  - press_comp = (press_comp + (var1 + var2 + var3 + (par_p7 * 128.0)) / 16.0) / 100

환경 센서

- par_p1 ~ par_t10: 보정 계수, press_adc: 저수준 측정 데이터, press_comp: 파스칼 단위 기압

Variable name	Register address (LSB/MSB)
par_p1	0x8E / 0x8F
par_p2	0x90 / 0x91
par_p3	0x92
par_p4	0x94 / 0x95
par_p5	0x96 / 0x97
par_p6	0x99
par_p7	0x98
par_p8	0x9C / 0x9D
par_p9	0x9E / 0x9F
par_p10	0xA0
press_adc	0x21<7:4> / 0x20 / 0x1F

환경 센서

- 보정 계수와 측정 결과로부터 상대 습도 계산
  - var1 = hum_adc - ((par_h1 * 16.0) + ((par_h3 / 2.0) * temp_comp))
  - var2 = var1 * ((par_h2 / 262144.0) * (1.0 + ((par_h4 / 16384.0) * temp_comp) + ((par_h5 / 1048576.0) * temp_comp * temp_comp)))
  - var3 = par_h6 / 16384.0
  - var4 = par_h7 / 2097152.0
  - hum_comp = var2 + ((var3 + (var4 * temp_comp)) * var2 * var2)

환경 센서

- par_h1 ~ h7: 보정 계수, hum_adc: 저수준 측정 데이터, hum_comp: 상대 습도

Variable name	Register address (LSB/MSB)
par_h1	0xE2<7:4> / 0xE3
par_h2	0xE2<7:4> / 0xE1
par_h3	0xE4
par_h4	0xE5
par_h5	0xE6
par_h6	0xE7
par_h7	0xE8
hum_adc	0x26 / 0x25

환경 센서

- 측정 결과로부터 저항 테이블을 이용해 유해 가스 검출량 계산
  - var1 = ((1340.0 + (5.0 * range_switching_error)) * (const_array1[gas_range])) / 65536
  - var2 = ((gas_adc * 32768) - 16777216) + var1
  - var3 = (const_array2[gas_range] * var1) / 512
  - gas_res = (var3 + (var2 / 2)) / var2
  - gas_range: 저항 테이블 인덱스, gas_adc: 저수준 측정 데이터
  - range_switching_error: 보정 계수, gas_res: 가스 검출량 따른 저항값

Variable name	Register address (LSB/MSB)
gas_adc	0x2B<7:6> / 0x2A
gas_range	0x2B<3:0>
range_switching_error	0x04<7:4>

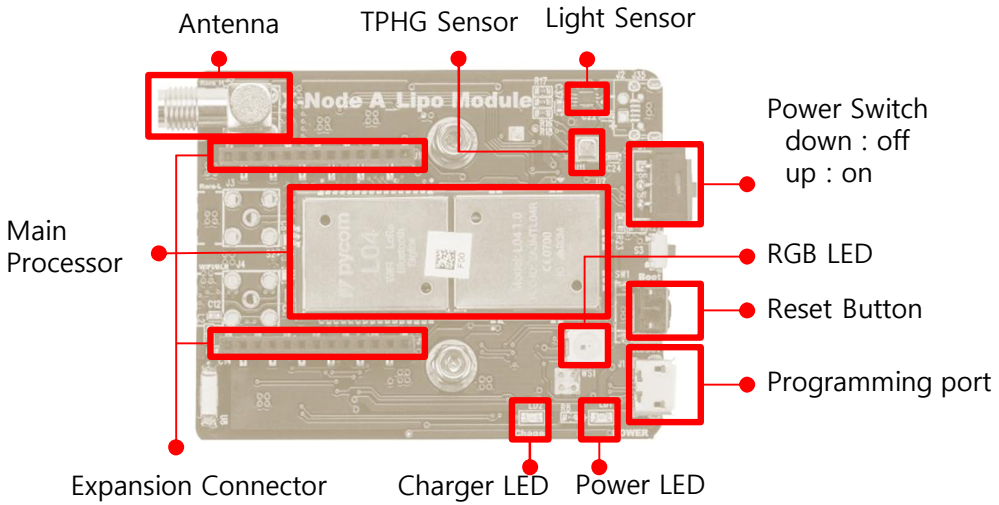
환경 센서

- const_array1, const_array2: 가스 저항 테이블

gas_range_r	Constants to be integrated into the driver			
	Floating point		Integer	
	const_array1	const_array2	const_array1_int	const_array2_int
0	1	8000000	2147483647	4096000000
1	1	4000000	2147483647	2048000000
2	1	2000000	2147483647	1024000000
3	1	1000000	2147483647	512000000
4	1	499500.4995	2147483647	255744255
5	0.99	248262.1648	2126008810	127110228
6	1	125000	2147483647	64000000
7	0.992	63004.03226	2130303777	32258064
8	1	31281.28128	2147483647	16016016
9	1	15625	2147483647	8000000
10	0.998	7812.5	2143188679	4000000
11	0.995	3906.25	2136746228	2000000
12	1	1953.125	2147483647	1000000
13	0.99	976.5625	2126008810	500000
14	1	488.28125	2147483647	250000
15	1	244.140625	2147483647	125000

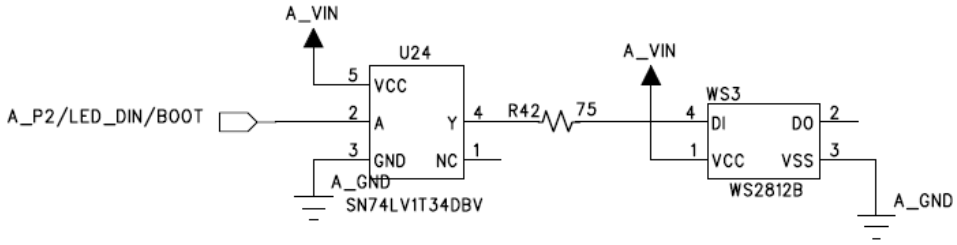
하드웨어 구성 – A Type

- Xnode A type에는 다음 센서가 기본 내장됨
  - RGB Led
    - 상태 알림용 Led
  - 배터리 전압 센서
    - 배터리 잔량 모니터링
  - 빛 센서
    - 주변 빛의 밝기 모니터링
  - 환경 센서
    - 공기 중 섭씨온도 및 상대 습도, 기압, 가스(에탄올, 유기화합물) 모니터링



RGB LED – A Type

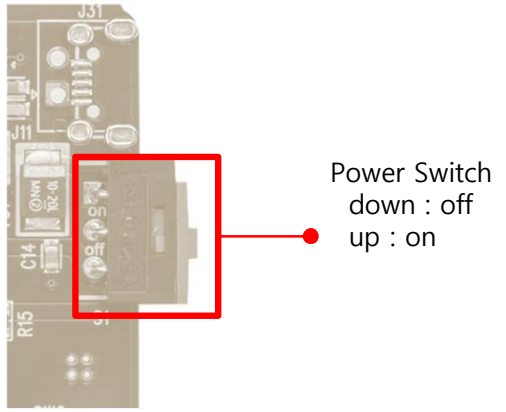
- RGB LED
  - XNode A Type의 P2 핀을 사용하여 RGB LED(WS2814)를 제어



XNode	device
P2	RGB LED

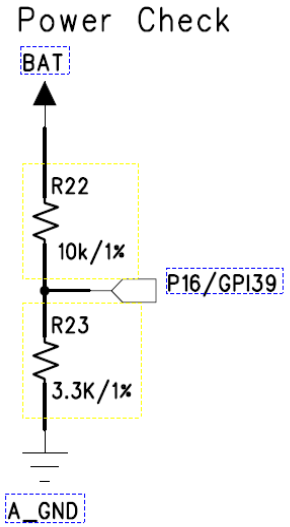
배터리 잔량 – A Type

- XNode 전원 공급
  - XNode는 배터리를 사용하여 전원 공급
  - 배터리는 USB 포트 기반으로 충전
  - 전원 스위치를 On 하여 사용
  - 배터리는 공칭전압 3.7V인 리튬폴리머 배터리를 사용



배터리 잔량 – A Type

- ADC로 배터리 전압을 측정해 현재 배터리 상태 인식
  - 배터리 전원 흐름 사이에 분배 저항을 구성한 후 프로세서의 P16에 연결됨
    - Vout = (R2 / (R1 + R2)) * Vin
  - R1 = 10K, R2 = 3.3K
    - Vin = 3.2V
      - Vout = (3.3 / (10 + 3.3)) * 3.2 = 0.793985V

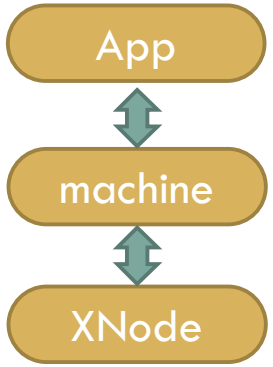


배터리 잔량 – A Type

- 배터리 전압은 전압 분배 회로에 의해 전압 강하된 상태로 인식
  - ADC는 측정 전압에 대한 디지털 변환 레벨을 반환하므로 ADC 참조 전압과 해상도를 이용해 전압으로 변환
    - XNode ADC의 참조 전압은 1.1V이고 해상도는 12bit (0~4095)
    - volt = (adc*1.1/4095)*((3.3+10)/3.3)
  - 배터리 전압에 따라 최대 측정값은 4.2V 이지만, 분배 저항과 배터리에 따라 오차가 있음
    - 최소 측정값이 3.2V 이하면 XNode 동작을 보장할 수 없으므로 배터리를 충전할 것

machine 라이브러리

- machine 라이브러리
  - XNode에 기본 내장됨
    - 하드웨어 제어 필요한 GPIO, I2C, ADC 제어 기능 제공
  - import machine
    - machine.Pin: 디지털 입출력인 GPIO 제어 클래스
    - machine.ADC: 아날로그 신호를 디지털로 변환하는 ADC 제어 클래스
    - machine.I2C: 임베디드 표준 시리얼 통신 중 하나인 I2C 제어 클래스



machine 라이브러리 – B type

- Pin class
  - Pin(id, mode, pull=-1, *, value=0) : Pin 객체 생성
    - id: GPIO 번호 설정. 문자열(대문자) 사용
    - mode: 핀 모드 설정. Pin.IN, Pin.OUT 중 하나
    - pull: 풀업 저항 설정. None, Pin.PULL_UP, Pin.PULL_DOWN 중 하나
    - value: Pin.OUT일 때 초기 출력 지정 (0 또는 1). 기본값은 0
  - Pin.value([x]) : 입력 또는 출력
    - x: 출력 지정으로 0 또는 1
      - Pin.IN일 때 생략하면 현재 상태 반환 (0 또는 1)

machine 라이브러리 – B type

- Pin.on(): Pin.OUT에 대해 HIGH 출력
- Pin.off(): Pin.OUT에 대해 LOW 출력
- Pin.mode([mode]): 현재 핀 모드를 읽거나 다시 설정
  - mode: 현재 핀 모드 설정, 생략하면 현재 핀 모드 반환
- Pin.pull([pull]): 현재 풀업 설정을 읽거나 다시 설정
  - pull: 현재 핀의 풀 업 설정. 생략하면 현재 풀업 반환

machine 라이브러리 – A type

- Pin class
  - Pin(id, mode=Pin.OUT, pull=None, alt) : Pin 객체 생성
    - id: GPIO 번호 설정. 문자열(대문자) 사용
    - mode: 핀 모드 설정. Pin.IN, Pin.OUT 중 하나
    - pull: 풀업 저항 설정. None, Pin.PULL_UP, Pin.PULL_DOWN 중 하나
    - alt: 대체 함수의 아이디
  - Pin.value([x]) : 입력 또는 출력
    - x: 출력 지정으로 0 또는 1
      - Pin.IN일 때 생략하면 현재 상태 반환 (0 또는 1)

machine 라이브러리 – A type

- Pin.mode([mode]): 현재 핀 모드를 읽거나 다시 설정
  - mode: 현재 핀 모드 설정, 생략하면 현재 핀 모드 반환
  - Pin.IN, PIN.OUT
- Pin.pull([pull]): 현재 풀업 설정을 읽거나 다시 설정
  - pull: 현재 핀의 풀 업 설정. 생략하면 현재 풀업 반환
  - Pin.PULL_UP, Pin.PULL_DOWN

machine 라이브러리 – B type

- ADC class
  - ADC(id) : ADC 객체 생성
    - id : 채널 번호. 문자열 사용
  - ADC.read() : ADC 변환 결과 읽기
    - 12bit (0~4095) 범위 값 반환
  - ADC.read_u16(): ADC 변환 결과 읽기
    - 16bit (0~65535) 범위 값 반환

machine 라이브러리 – A type

- ADC class
  - ADC() : ADC 객체 생성
  - ADC.channel (pin) : analog pin 생성
    - pin : 핀 번호 문자열로 사용
  - ADC.value() : ADC 변환 결과 읽기
    - 12bit (0~4095) 범위 값 반환

machine 라이브러리 – B type

- I2C class
  - I2C(id, *, freq=400000) : I2C 객체 생성
    - id : 버스 선택. XNode는 1로 고정
    - freq : 전송 속도 설정. 기본값은 400000
  - I2C.scan() : I2C 장치 검색
    - 버스에서 검색한 장치 주소 반환. 유효 범위는 0x08~0x77

machine 라이브러리 – B type

- I2C.readfrom (addr, nbytes, stop=True) : 데이터 읽기
  - addr: 장치 주소
  - nbytes: 읽을 바이트 수
  - stop: True이면 종료 시 STOP 조건 생성
  - bytes 타입으로 읽은 데이터 반환
- I2C.writeto (addr, buf, stop=True) : 데이터 쓰기
  - addr: 장치 주소
  - buf : 쓸 데이터가 포함된 bytes 타입 버퍼
    - 바이트를 쓴 후 NACK가 수신되면 나머지 바이트는 전송되지 않음
  - stop : True이면 종시 시 STOP 조건 생성
  - 수신된 ACK 횟수 반환

machine 라이브러리 – B type

- I2C.readfrom_mem(addr, memaddr, nbytes): 메모리 데이터 읽기
  - addr: 장치 주소
  - memaddr: 장치의 해당 메모리(레지스터) 주소
  - nbytes : 읽을 바이트 수
  - bytes 타입으로 읽은 데이터 반환

machine 라이브러리 – B type

- I2C.writeto_mem(addr, memaddr, buf): 메모리에 데이터 쓰기
  - addr: 장치 주소
  - memaddr :장치의 해당 메모리(레지스터) 주소
  - buf : 쓸 데이터가 포함된 bytearray 타입 버퍼

machine 라이브러리 – A type

- I2C class
  - I2C(id, *, baudrate=100000) : I2C 객체 생성
    - id : 버스 선택. XNode는 1로 고정
    - baudrate: 전송 속도 설정. 기본값은 100000
  - I2C.scan() : I2C 장치 검색
    - 버스에서 검색한 장치 주소 반환. 유효 범위는 0x08~0x77

machine 라이브러리 – A type

- I2C.readfrom (addr, nbytes) : 데이터 읽기
  - addr: 장치 주소
  - nbytes: 읽을 바이트 수
  - bytes 타입으로 읽은 데이터 반환
- I2C.writeto (addr, buf) : 데이터 쓰기
  - addr: 장치 주소
  - buf : 쓸 데이터가 포함된 bytes 타입 버퍼
    - 바이트를 쓴 후 NACK가 수신되면 나머지 바이트는 전송되지 않음
  - 수신된 ACK 횟수 반환

machine 라이브러리 – A type

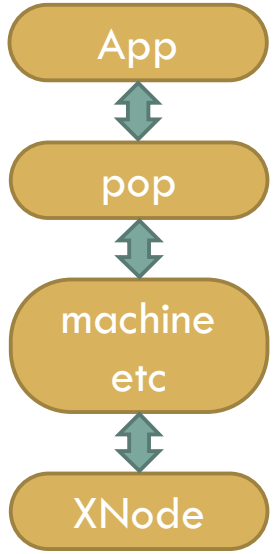
- I2C.readfrom_mem(addr, memaddr, nbytes): 메모리 데이터 읽기
  - addr: 장치 주소
  - memaddr: 장치의 해당 메모리(레지스터) 주소
  - nbytes : 읽을 바이트 수
  - bytes 타입으로 읽은 데이터 반환

machine 라이브러리 – A type

- I2C.writeto_mem(addr, memaddr, buf): 메모리에 데이터 쓰기
  - addr: 장치 주소
  - memaddr :장치의 해당 메모리(레지스터) 주소
  - buf : 쓸 데이터가 포함된 bytearray 타입 버퍼

Pop 라이브러리

- Pop 라이브러리
  - 한백전자에서 제공하는 확장 라이브러리
    - machine과 같이 XNode에 기본 내장 라이브러리로 구현
    - 시스템을 설계할 때 아이디어 완성에 집중할 수 있음
  - XNode에 포함된 기본 센서 및 확장 모듈별로 구분해 제공
    - BASIC: Buzzer class, Leds class, Buttons class
    - IRTHERMO: IRThermo class
    - PIR: Pir class
    - IMU: IMU class
    - GPS: GPS class



Pop 라이브러리

- MicroPython의 메모리 제한으로 기본 내장되지 않음
  - 확장 모듈에 따라 별도 파일로 구현
    - XNode의 /flash/lib 경로에 해당 파일을 복사한 후 사용

Pop 라이브러리

- 기능에 따른 적용 라이브러리

구분	Sensor/Actuator	location
XNode 기본 센서	LED 배터리 잔량 빛 센서 환경 센서(온도,기압,습도,가스)	USB > Library > CORE > lib > pop.py, core_a.py, core_b.py
BASIC 확장 모듈	버튼 2개 LED 8개 부저 1개	USB > Library > EXT > lib > BASIC.py
PIR 확장 모듈	적외선 기반 움직임 감지 센서	USB > Library > EXT > lib > PIR.py
IRTHERMO 확장 모듈	비접촉 온도 센서	USB > Library > EXT > lib > IRTHERMO.py
IMU 확장 모듈	9축 IMU 센서	USB > Library > EXT > lib > IMU.py

Pop 라이브러리 -기본 센서

- Led Class(Xnode B Type only)
  - Led(): Led 객체 생성
  - Led.on(): Led 켜기
  - Led.off(): Led 끄기
  - Led.stat(): Led 현재 상태 반환. Led가 꺼져 있으면 False, 켜져 있으면 True 반환

Pop 라이브러리 -기본 센서

- RgbLed class (Xnode A Type only)
  - RgbLed(): Led 객체 생성
  - RgbLed.on(): setColor() 함수로 설정된 색상으로 RgbLed 켜기
  - RgbLed.off(): RgbLed 끄기
  - RgbLed.setColor([color]): 출력할 색상 설정
    - color : 0xRRGGBB의 24비트 값.
      - ex. 0xFF0000 : 적색, 0x00FF00 : 녹색, 0x0000FF : 청색
  - RgbLed.getColor(): 현재 설정된 색상 반환
  - RgbLed.stat(): RgbLed 현재 상태 반환. RgbLed가 off 상태이면 False, on 상태면 True 반환

Pop 라이브러리 -기본 센서

- Battery Class
  - Battery(): 배터리 잔량 측정 객체 생성
  - Battery.read(): 배터리 잔량 읽기
    - 실제 배터리 전압 반환. 3.2V 이하면 배터리 교체 필요

Pop 라이브러리 -기본 센서

- Light Class
  - Light(cont=False): Light 객체 생성
    - cont: True이면 연속 측정 모드 사용. 기본값은 False
  - Light.read(): 빛 센서를 통해 주변 밝기 읽기
    - lux 단위 조도 값 반환
  - Light.stop(): 연속 측정 모드일 때 측정 중단
- Tphg Class
  - Tphg(): 온도, 기압, 습도, 가스 측정 객체 생성
  - Tphg.read(): 온도, 기압, 습도, 가스 읽기
    - 반환값은 튜플 객체로 (온도, 기압, 습도, 가스)

Pop 라이브러리 -기본 센서

- Tphg.sealevel(altitude): 현재 고도와 기압을 기준으로 현재 해면 기압 계산
  - altitude: 현재 측정 고도
  - 반환값은 튜플 객체로 (해면 기압, 기압)
- Tphg.altitude(sealevel): 현재 해면 기압과 기압을 기준으로 현재 고도 계산
  - sealevel: 현재 측정 기준 해면 기압
  - 반환값은 튜플 객체로 (고도, 기압)

기본 센서 제어

- 준비물

준비물	
1	PC
2	XNode 1ea
3	USB cable 1ea
4	XNode 제공 USB 메모리 (D: 로 가정)

  - XNode와 PC를 USB 케이블로 연결
  - (옵션) 다음 명령으로 XNode 파일 시스템 초기화
    - XNode B Type : xnode -p <포트> format b
    - XNode A Type : xnode -p <포트> format a
  - D:₩Library₩CORE를 작업 폴더인 C:₩XNode로 복사
    - xnode -p <포트> put D:₩Library₩CORE₩lib

기본 센서 제어

- VS Code 통합 개발환경 실행
  - 작업 폴더인 C:₩XNode₩CORE 열기
  - 터미널 창 실행 후 XNode 탐색
    - xnode scan
  - XNode를 초기화했다면 C:₩XNode₩CORE 아래 lib 폴더 내용을 XNode로 복사
    - xnode -p <포트> put lib
  - app.py를 열고 기본 센서 제어 코드 구현
    - XNode에 설치하지 않고 실행만 한다면 파일명 변경 가능
  - 구현한 마이크로파이썬 파일을 XNode에서 실행
    - xnode -p <포트> run <파일명>.py

지연 시간

- 센서 제어는 프로세서와 센서 사이 시간 동기가 중요
  - 프로세서의 빠른 처리 시간을 느린 센서 처리 시간에 맞춰 지연
  - time 모듈에서 제공하는 함수 사용
    - time.sleep(n): 초 단위 지연
      - n: 초 단위 지연 시간. 소수점을 사용하면 밀리초 단위
    - time.sleep_ms(n): 밀리초 단위 지연
      - n: 밀리초 단위 지연 시간
    - time.ticks_ms(): XNode가 부팅된 후부터 밀리초 단위로 카운트한 횟수
      - 정수 단위 카운트 횟수 반환
    - time.ticks_diff(m, n): 두 시차를 계산해 반환
      - m, n: 정수 단위 카운트 횟수

지연 시간

- sleep() 함수 지연 성능 측정
  - 마이크로파이썬은 밀리초 단위 실행 오차가 있음

```
01: import time
02:
03: start = time.ticks_ms()
04:
05: time.sleep(1)
06: time.sleep_ms(500)
07:
08: delta = time.ticks_diff(time.ticks_ms(), start)
09:
10: print("Operation took %d ms to execute"%(delta))
```

```
PS C:\XNode\CORE> xnode -p com3 run core_time.py
Operation took 1507 ms to execute
PS C:\XNode\CORE> xnode -p com3 run core_time.py
Operation took 1502 ms to execute
PS C:\XNode\CORE> xnode -p com3 run core_time.py
Operation took 1507 ms to execute
```

LED 켜고 끄기 - Xnode B Type

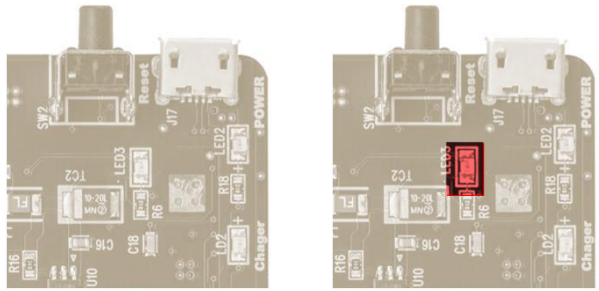
- Pop의 Led 클래스로 LED 켜고 끄기
  - /flash/lib/에서 pop 모듈의 Led 클래스 로드
    - from pop import Led
  - 내장된 time 모듈 로드
    - import time
  - Led 객체 생성
    - l = Led()
  - Led 켜고 끄기
    - l.on()
    - time.sleep(3) #3초 대기
    - l.off()

LED 제어 예제 1

- LED 깜빡이기

```
01: from pop import Led
02: import time
03:
04: l = Led()
05:
06: for _ in range(10):
07:     l.on()
08:     print("Led on")
09:     time.sleep(.1)
10:     l.off()
11:     print("Led off")
12:     time.sleep(.1)
```

```
PS C:\XNode\CORE> xnode -p com3 run core_led_loop.py
Led on
Led off
Led on
Led off
Led on
```



LED 제어 예제 2

- 모스 부호를 LED로 출력하는 mos(ch, dot=0.2) 함수 구현
  - ch: 영문자(대문자)
  - dot: '.' 문자 지연시간. '-' 문자 지연시간은 dot의 3배

```
01: mos_table = {
02:     'A':".-", 'B':"-...", 'C':"-.-.", 'D':"-..", 'E':".", 'F':"..-.", 'G':"--.", 'H':"....", 'I':"..", 'J':".---", 'K':"-.-", 'L':".-..", 'M':"--", 'N':"-.",
03:     'O':"---", 'P':".--.", 'Q':"--.-", 'R':".-.", 'S':"...", 'T':"-", 'U':"..-", 'V':"...-", 'W':".--", 'X':"-..-", 'Y':"-.--", 'Z':"--.."
04: }
05: def mos(ch, dot=0.2):
06:     for m in mos_table[ch]:
07:         l.on()
08:         time.sleep(dot) if m == '.' else time.sleep(dot*3)
09:         l.off()
10:         time.sleep(dot)
```

LED 제어 예제 2

- 앞서 구현한 mos() 함수로 "SOS" 출력

```
01: from pop import Led
02: import time
03:
04: l = Led()
05:
06: mos_table = {
07:     ...
08: }
09:
10: def mos(ch, dot=0.2):
11:     ...
12:
13: print('S'); mos('S')
14: print('O'); mos('O')
15: print('S'); mos('S')
```

```
PS C:\XNode\CORE> xnode -p com3 run core_led_mos.py
S
O
S
```

LED 제어 예제 3

- Led 클래스를 상속한 LedEx 클래스를 만든 후 toggle() 메소드 추가
  - Led 클래스에서 제공하는 on(), off() 사용

```
01: class LedEx(Led):
02:     def __init__(self):
03:         super().__init__()
04:         self.stat = False
05:
06:     def toggle(self):
07:         self.stat = not self.stat
08:         self.on() if self.stat else self.off()
```

LED 제어 예제 3

- LedEx 클래스의 stat 속성과 toggle() 메소드를 이용해 LED 반전

```
01: from pop import Led
02: import time
03:
04: class LedEx(Led):
05:     ...
12:
13: l = LedEx()
14:
15: for _ in range(10):
16:     l.toggle()
17:     print("Led on" if l.stat else "Led off")
18:     time.sleep(.1)
```

```
PS C:\XNode\CORE> xnode -p com3 run core_led_class.py
Led on
Led off
Led on
Led off
Led on
```

LED 제어 예제 4

- machine의 Pin 클래스로 LED 켜고 끄기
  - Pop의 Led 클래스는 machine의 Pin 클래스로 구현됨
  - machine 라이브러리의 Pin 클래스 로드
    - from machine import Pin
  - LED가 연결된 GPIO D9 핀을 인자로 Pin 객체 생성
    - 출력으로 지정한 후 초깃값으로 1 출력 -> 꺼짐 유지 (Active Low)
    - led = Pin("D9", Pin.OUT, value=1)
  - 현재 LED 핀 상태 읽기
    - print(led.value())
  - LED 핀에 LOW 출력
    - led.value(1)
  - LED 핀에 HIGH 출력
    - led.value(0)

LED 제어 예제 4

- 소프트웨어 PWM으로 밝기 조절
  - 50Hz 주기로 LED 켜고, 끄기를 반복할 때 켜는 시간을 조절해 평균 전압을 낮춤

```
01: from machine import Pin
02: import time
03:
04: led = Pin("D9", Pin.OUT, value=1)
05: hz50 = 1/50
06:
07: led.value(0)
08: time.sleep(2)
09:
10: for i in range(1, 50*2+1): #2초간 밝기를 1/2로 줄임
11:         led.value(0)
12:         print(led.value(), end='')
13:         time.sleep(int(hz50/2))
14:         led.value(1)
15:         time.sleep(int(hz50/2))
16:         print(led.value(), end='') if i % 20 else print(led.value())
```

```
PS C:\XNode\CORE> xnode -p com3 run core_led_raw_swpwm.py
01010101010101010101010101010101010101010101
01010101010101010101010101010101010101010101
01010101010101010101010101010101010101010101
01010101010101010101010101010101010101010101
01010101010101010101010101010101010101010101
```

RgbLED 켜고 끄기 - Xnode A Type

- Pop의 RgbLed 클래스로 RgbLed 켜고 끄기
  - /flash/lib/에서 pop 모듈의 RgbLed 클래스 로드
    - from pop import RgbLed
  - 내장된 time 모듈 로드
    - import time
  - RgbLed 객체 생성
    - l = RgbLed()
  - 출력할 색상 설정
    - l.setColor(0xffffff)

RgbLED 켜고 끄기 - Xnode A Type

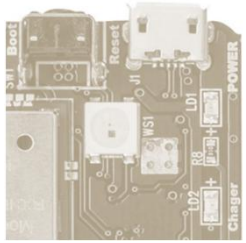
- Pop의 RgbLed 클래스로 RgbLed 켜고 끄기
  - RgbLed켜고 끄기
    - l.on()
    - time.sleep(3) #3초 대기
    - l.off()

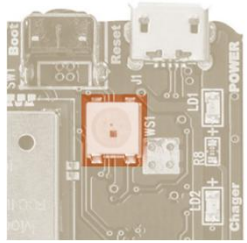
RgbLED 제어 예제 1

- RgbLED 흰색으로 깜빡이기

```
01: from pop import RgbLed
02: from pop import time
03:
04: l = RgbLed()
05: l.setColor(0xFFFFFF)
06:
07: l.on()
08: print("Led on")
09: time.sleep(3)
10: l.off()
11: print("Led off")
```

```
PS C:\XNode\CORE> xnode -p com3 run core_rgbled.py
Led on
Led off
```





RgbLED 제어 예제 2

- RgbLED 난수 색상 출력

```
01: from pop import RgbLed, rand
02: from pop import time
03:
04: l = RgbLed()
05:
06: for i in range(10):
07:     l.setColor(rand()>>8)
08:     print("setColor : 0x%x"%(l.getColor()))
09:     l.on()
10:     time.sleep(.5)
11:
12: l.off()
13: print("Led off")
```

```
PS C:\XNode\CORE> xnode -p com3 core_rgbled_rand.py
setColor : 0x1c8ec7
setColor : 0x71b8dc
setColor : 0xb7db6d
setColor : 0x9bcde6
setColor : 0xf97c3e
setColor : 0x8f47a3
setColor : 0x68349a
setColor : 0x2693c9
setColor : 0xf279bc
setColor : 0xaf57ab
Led off
```

배터리 잔량 측정

- 배터리 잔량
  - XNode 공칭전압 3.7인 배터리를 사용
  - 배터리 잔량이 3.2V 이하면 XNode의 동작을 보장할 수 없음
    - USB 케이블을 연결해 배터리 충전
  - Pop의 Battery 클래스로 배터리 잔량을 측정 가능

배터리 잔량 측정

- Pop의 Battery 클래스로 배터리 잔량 측정
  - /flash/lib/에서 pop 모듈의 Battery 클래스 로드
    - from pop import Battery
  - Battery 객체 생성
    - b = Battery()
  - Battery 상태 읽기
    - print(b.read())

배터리 잔량 측정 예제 1

- 현재 배터리 상태 측정

```
01: from pop import Battery
02: from pop import time
03:
04: b = Battery()
05:
06: for _ in range(5):
07:     print("Battery: %.1f Volt"%(b.read()))
08:     time.sleep(1)
```

```
PS C:\XNode\CORE> xnode -p com3 run core_battery.py
Battery: 4.1 Volt
Battery: 4.1 Volt
Battery: 4.1 Volt
Battery: 4.1 Volt
Battery: 4.1 Volt
```

배터리 잔량 측정 예제 2

- 구간별 배터리 상태 표시. (주의: 실제 배터리 상태 변화는 긴 시간이 필요함)
  - 3개 구간(100% ~ 80%, 79% ~ 30%, 30% 미만)으로 구분해 "HIGH", "MIDDLE", "LOW" 출력

```
01: from pop import Battery
02: import time
03:
04: b = Battery()
05:
06: max = 4.2
07: min = 3.2
08: block = 10
09:
10: step = (max-min) / block
11: block_table = {i*10:(min+step*i) for i in range(block+1)}
12:
13: for _ in range(10):
14:     val = b.read()
15:     print("Battery: %.1f Volt"%(val), end=", ")
16:     print("HIGH") if val >= block_table[80] else print("MIDDLE") if val >= block_table[30] else print("LOW")
17:     time.sleep(1)
```

배터리 잔량 측정 예제 2

- 결과

```
PS C:\XNode\CORE> xnode -p com3 run core_battery_range.py
Battery: 3.8 Volt, HIGH
Battery: 3.9 Volt, HIGH
Battery: 3.6 Volt, HIGH
Battery: 3.6 Volt, HIGH
Battery: 3.5 Volt, HIGH
Battery: 3.3 Volt, MIDDLE
Battery: 3.3 Volt, MIDDLE
Battery: 3.0 Volt, LOW
Battery: 3.0 Volt, LOW
Battery: 2.9 Volt, LOW
```

배터리 잔량 측정 예제 3 – B type

- machine의 ADC 클래스로 배터리 잔량 측정
  - Pop의 Battery 클래스는 machine의 ADC 클래스로 구현됨
  - machine 라이브러리의 ADC 클래스 로드
    - from machine import ADC
  - 배터리 측정 핀이 연결된 GPIO D2를 인자로 ADC 객체 생성
    - adc = ADC("D2")
  - ADC에서 읽은 측정값을 ADC 참조 전압과 해상도를 적용해 측정 전압으로 변환
    - raw = adc.read()
    - conv = round(raw * 3.3 / 4095, 1)
    - battery = conv * 1.6

배터리 잔량 측정 예제 3 – B type

```
01: from machine import ADC
02: import time
03:
04: adc = ADC("D2")
05:
06: for _ in range(10):
07:     raw = adc.read()
08:     conv = round(raw * 3.3 / 4095, 1)
09:     battery = conv * 1.6
10:     print("raw = %d, conv = %.1f, battery = %.1f"%(raw, conv, battery), end=', ')
11:
12:     time.sleep(1)
```

```
PS C:\XNode\CORE> xnode -p com3 run core_battery_raw.py
raw = 3214, conv = 2.6, battery = 4.2
raw = 3214, conv = 2.6, battery = 4.2
```

배터리 잔량 측정 예제 4 – A type

- machine의 ADC 클래스로 배터리 잔량 측정
  - Pop의 Battery 클래스는 machine의 ADC 클래스로 구현됨
  - machine 라이브러리의 ADC 클래스 로드
    - from machine import ADC
  - 배터리 측정 핀이 연결된 GPIO ‘P16’을 인자로 ADC 객체 생성
    - _adc = ADC()
    - adc = _adc.channel(pin='P16')
  - ADC에서 읽은 측정값을 ADC 참조 전압과 해상도를 적용해 측정 전압으로 변환
    - raw = adc.read()
    - conv = round(raw * 1.1 / 4095, 1)
    - battery = conv * (13.3/3.3)

배터리 잔량 측정 예제 4 – A type

```
01: from machine import ADC
02: import time
03:
04: _adc = ADC()
05: adc = _adc.channel(pin='P16')
06:
07: for _ in range(10):
08:     raw = adc.value()
09:     conv = round(raw * 1.1 / 4095, 1)
10:     battery = conv * (13.3/3.3)
11:     print("raw = %d, conv = %.1f, battery = %.1f"%(raw, conv, battery))
12:
13:     time.sleep(1)
```

```
PS C:\XNode\CORE> xnode -p com3 run core_battery_raw.py
raw = 3214, conv = 2.6, battery = 4.2
raw = 3214, conv = 2.6, battery = 4.2
```

주변 밝기 측정

- Pop의 Light 클래스로 주변 밝기 읽기
  - /flash/lib/에서 pop 모듈의 Light 클래스 로드
    - from pop import Light
  - Light 객체 생성
    - l = Light()
  - 빛 센서값 읽기
    - print(l.read())

주변 밝기 측정 예제 1

- 센서에 도달한 빛의 양으로 주변 밝기 측정
  - 센서 표면을 손으로 가려 변화 파악

```
01: from pop import Light
02: import time
03:
04: l = Light()
05:
06: for _ in range(10):
07:     val = l.read()
08:     print("light = %d lx"%(val))
09:     time.sleep(1)
```

```
PS C:\XNode\CORE> xnode -p com3 run core_light.py
light = 251 lx
light = 115 lx
light = 14 lx
light = 9 lx
light = 244 lx
```

주변 밝기 측정 예제 2

- 연속 측정 모드 사용
  - 측정을 종료하려면 stop() 호출

```
01: from pop import Light
02: import time
03:
04: l = Light(True)
05:
06: for _ in range(5):
07:     val = l.read()
08:     print("light = %d lx"%(val))
09:     time.sleep(1)
10: l.stop()
```

```
PS C:\XNode\CORE> xnode -p com3 run core_light_cont.py
light = 252 lx
light = 252 lx
light = 42 lx
light = 21 lx
light = 249 lx
```

주변 밝기 측정 예제 3

- 주변 밝기와 KS 기준 조도 분류

활동 유형	분류	조도 범위	조명방법
어두운 분위기 중의 시식별 작업장	A	3-4-6	공간의 전반조명
어두운 분기기의 이용이 빈번하지 않은 장소	B	6-10-15	
어두운 분위기의 공공 장소	C	15-20-30	
잠시 동안의 단순 작업장	D	30-40-60	
시작업이 빈번하지 않은 작업장	E	60-100-150	
고휘도 대비 혹은 큰 물체 대상의 시작업 수행	F	150-200-300	작업면 조명
일반휘도 대비 혹은 작은 물체 대상의 시작업 수행	G	300-400-600	
저휘도 대비 혹은 매우 작은 물체 대상의 시작업 수행	H	600-1000-1500	
비교적 장시간 동안 저휘도 대비 혹은 매우 작은 물체 대상의 시작업 수행	I	1500-2000-3000	전반조명과 국부조명을 병행한 작업면 조명
장시간 동안 힘든 시작업 수행	J	3000-4000-6000	
희도 대비가 거의 안되며 작은 물체의 매우 특별한 시작업 수행	K	6000-10000-15000	

주변 밝기 측정 예제 3

- 주변 밝기에 따른 KS 기준 조도 분류 등급 출력

```
01: from pop import Light
02: import time
03:
04: l = Light()
05:
06: for _ in range(10):
07:     val = l.read()
08:     ret = None
09
10:     if val >= 3 and val <= 6:
11:         ret = 'A'
12:     elif val > 6 and val <= 15:
13:         ret = 'B'
14:     elif val > 15 and val <= 30:
15:         ret = 'C'
16:     elif val > 30 and val <= 60:
17:         ret = 'D'
```

주변 밝기 측정 예제 3

- (계속)

```
18:    elif val > 60 and val <= 150:
19:        ret = 'E'
20:    elif val > 150 and val <= 300:
21:        ret = 'F'
22:    elif val > 300 and val <= 600:
23:        ret = 'G'
24:    elif val > 600 and val <= 1500:
25:        ret = 'H'
26:    elif val > 1500 and val <= 3000:
27:        ret = 'I'
28:    elif val > 3000 and val <= 6000:
29:        ret = 'J'
30:    elif val > 6000 and val <= 15000:
31:        ret = 'K'
32:
33:    print("light = %d lx, level = %s"%(val, ret))
34:    time.sleep(1)
```

```
PS C:\XNode\CORE> xnode -p com3 run core_light_ks.py
light = 251 lx, level = F
light = 172 lx, level = F
light = 2 lx, level = None
light = 175 lx, level = F
light = 40 lx, level = D
light = 33 lx, level = D
light = 13 lx, level = B
light = 12 lx, level = B
light = 24 lx, level = C
light = 239 lx, level = F
```

주변 밝기 측정 예제 3

- 주변 밝기와 KS 기준 조도 분류를 딕셔너리와 for 루프로 수정

```
01: from pop import Light
02: import time
03:
04: ks_light = {'A':(3, 6),'B':(6, 15), 'C':(15, 30), 'D':(30, 60), 'E':(60, 150), 'F':(150, 300),
05:                 'G':(300, 600), 'H':(600, 1500), 'I':(1500, 3000), 'J':(3000, 6000), 'K':(6000, 15000)}
06: l = Light()
07:
08: for _ in range(10):
09:     val = l.read()
10:     ret = None
11:
12:     for k, v in ks_light.items():
13:         if val >= v[0] and val <= v[1]:
14:             ret = k
15:
16:     print("light = %d lx, level = %s"%(val, ret))
17:     time.sleep(1)
```

```
PS C:\XNode\CORE> xnode -p com3 run core_light_ks_dict.py
light = 277 lx, level = F
light = 271 lx, level = F
light = 75 lx, level = E
light = 269 lx, level = F
light = 177 lx, level = F
light = 158 lx, level = F
light = 268 lx, level = F
light = 38 lx, level = D
light = 6 lx, level = B
light = 2 lx, level = None
```

주변 밝기 측정 예제 4

- machine의 I2C 클래스로 주변 밝기 측정
  - Pop의 Light 클래스는 machine의 I2C 클래스로 구현됨
  - machine 라이브러리의 I2C 클래스와 time 모듈 로드
    - from machine import I2C
    - import time
  - 빛 센서가 연결된 I2C 버스 1을 인자로 I2C 객체 생성
    - i2c = I2C (1)
  - 초기화 절차 진행
    - I2c.writeto(0x23, bytes([0x00])
    - I2c.writeto(0x23, bytes([0x01])
    - I2c.writeto(0x23, bytes([0x])

주변 밝기 측정 예제 4

- 연속 고해상도 모드2로 측정 시작
  - I2c.writeto(0x23, bytes([0x10])
  - time.sleep(0.180)
- 빛 센서로부터 측정 결과 읽기
  - data = i2c.readfrom(0x23, 2)
- 읽은 2개의 데이터를 하나로 합친 후 룩스 값으로 변환
  - light = (data[0] << 8) | data[1]) / (1.2 * 2)
- 결과 출력
  - print(light)

주변 밝기 측정 예제 4

- 연속 모드로 평균 조도 측정

```
01: from machine import I2C
02: import time
03:
04: i2c = I2C(1)
05: i2c.writeto(0x23, bytes([0x01])) #power on
06: i2c.writeto(0x23, bytes([0x07])) #reset
07:
08: i2c.writeto(0x23, bytes([0x11])) #cont_hires_2 mode
09: time.sleep(0.180)
10:
11: light = 0
12: for i in range(1, 100+1):
13:     high, low = i2c.readfrom(0x23, 2)
14:     light += round((high << 8 | low) / (1.2 * 2))
15:     if not i % 10
16:         print("light = %d lx"%(light // 10))
17:         light = 0
18:     time.sleep(.1)
19: i2c.writeto(0x23, bytes([0x00])) #power down
```

```
PS C:\XNode\CORE> xnode -p com3 run core_light_raw.py
light = 238 lx
light = 221 lx
light = 216 lx
light = 175 lx
light = 89 lx
light = 89 lx
light = 200 lx
light = 45 lx
light = 236 lx
light = 221 lx
```

환경 센싱

- Pop의 Tphg 클래스로 환경 센싱
  - /flash/lib/에서 pop 모듈의 Tphg 클래스 로드
    - from pop import Tphg
  - Tphg 객체 생성
    - tphg = Light()
  - 빛 센서값 읽기
    - print(tphg.read())

주변 환경 센싱 예제 1

- 주변 환경 센싱(섭씨온도, 기압, 상대 습도, 유해 가스양)
  - 유해 가스양은 저항값 변화로 값이 작을수록 유해 가스가 많이 검출됨

```
01: from pop import Tphg
02: import time
03:
04: tphg = Tphg()
05:
06: for _ in range(5):
07:     val = tphg.read()
08:     print("Temp = %.1f C, Press = %.1f hPa, Humi = %.1f RH, Gas = %d ohm"%(val))
09:     time.sleep(1)
```

```
PS C:\XNode\CORE> xnode -p com3 run core_tphg.py
Temp = 24.6 C, Press = 1015.0 hPa, Humi = 30.6 RH, Gas = 37064 ohm
Temp = 24.6 C, Press = 1015.0 hPa, Humi = 30.6 RH, Gas = 17065 ohm
Temp = 24.7 C, Press = 1015.0 hPa, Humi = 30.6 RH, Gas = 20648 ohm
Temp = 24.7 C, Press = 1015.0 hPa, Humi = 30.6 RH, Gas = 25061 ohm
Temp = 24.8 C, Press = 1015.0 hPa, Humi = 30.6 RH, Gas = 28273 ohm
```

주변 환경 센싱 예제 2

- 온도, 습도 읽기
  - Tphg.read()가 반환하는 튜플 요소를 풀어서 받으면 인덱스를 사용할 필요가 없음

```
01: from pop import Tphg
02: import time
03:
04: tphg = Tphg()
05:
06: for _ in range(5):
07:     temp, _, humi, _ = tphg.read()
08:     print("Temp = %.1f C, Humi = %.1f RH"%(temp, humi))
09:     time.sleep(1)
```

```
PS C:\XNode\CORE> xnode -p com3 run core_tphg_th.py
Temp = 24.4 C, Humi = 31.3 RH
Temp = 24.4 C, Humi = 31.3 RH
Temp = 24.5 C, Humi = 31.2 RH
Temp = 24.5 C, Humi = 31.2 RH
Temp = 24.5 C, Humi = 31.1 RH
```

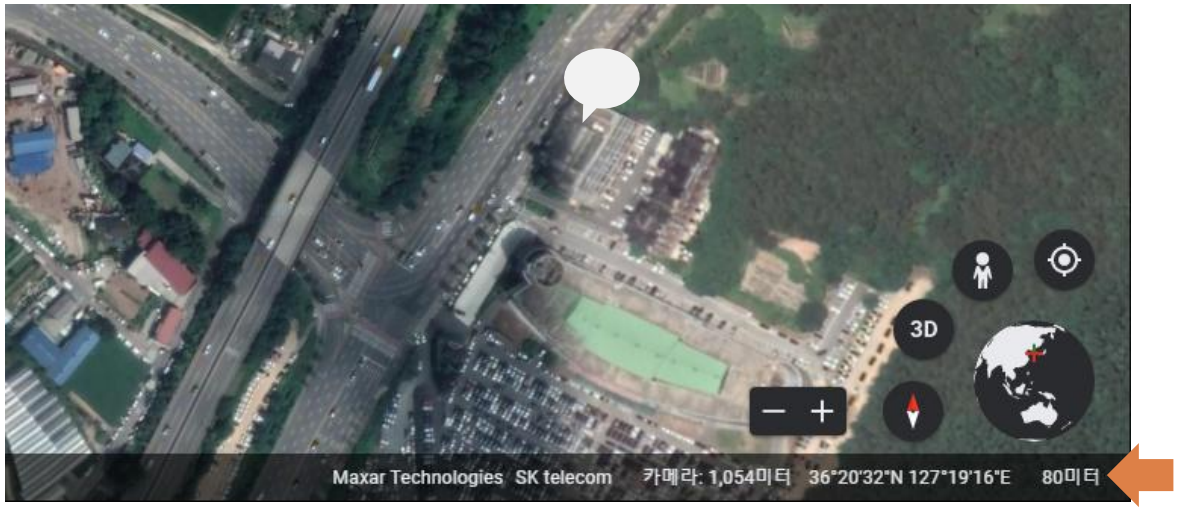
주변 환경 센싱 예제 3

- 해면 기압(해수면의 압력) 계산
  - 측정 위치의 기압 P와 고도 altitude를 이용해 hPa 단위 해면 기압 계산

$$p_0 = \frac{p}{\left(1 - \frac{altitude}{44330}\right)^{5.255}}$$

주변 환경 센싱 예제 3

- 현재 위치의 고도는 구글 어스를 통해 확인 가능
  - https://earth.google.com/web/
  - 왼쪽에서 검색 클릭 > 오른쪽 하단에 해당 장소의 고도가 표시됨 (마우스를 이동할 때마다 바뀜)



주변 환경 센싱 예제 3

- 현재 고도를 기준으로 해면 기압 계산
  - 예) 대전 유성구 유성대로 518 (고도 80m) 5층(+10m) 고도 90m

```
01: from pop import Tphg
02: import time
03:
04: tphg = Tphg()
05:
06: for _ in range(5):
07:     sea_level, press = tphg.sealevel(90)
08:     print("Sea Level = %.1f hPa, Pressure = %d"%(sea_level, press))
09:     time.sleep(1)
```

```
PS C:\XNode\CORE> xnode -p com3 run core_tphg_sealevel.py
Sea Level = 1029.2 hPa, Pressure = 1018 hPa
Sea Level = 1029.2 hPa, Pressure = 1018 hPa
Sea Level = 1029.2 hPa, Pressure = 1018 hPa
Sea Level = 1029.2 hPa, Pressure = 1018 hPa
Sea Level = 1029.3 hPa, Pressure = 1018 hPa
```

주변 환경 센싱 예제 4

- 절대 고도 계산
  - 측정 위치의 기압 P와 해면 기압 P0 이용 미터 단위 고도 계산

$$altitude = 44330 * \left(1 - \left(\frac{p}{p_0}\right)^{\frac{1}{5.255}}\right)$$

주변 환경 센싱 예제 4

- 현재 해면 기압은 기상청에서 확인 가능
  - https://www.weather.go.kr/weather/observation/currentweather.jsp
  - 선택 > 해면 기압, 지역 > 대전,세종,충청도

기상실황표 2020.11.10.14:00

지점	03H	06H	09H	12H	15H	18H	21H	24H
금산	1027.1	1028.1	1029.7	1029.1				
대전	1027.4	1027.9	1029.5	1029.1				
보령	1026.1	1026.7	1028.2	1028.4				

주변 환경 센싱 예제 4

- 해면 기압을 기준으로 현재 고도 계산
  - 예) 2020.11.10 대전 해면 기압 1029.1

```
01: from pop import Tphg
02: import time
03:
04: tphg = Tphg()
05:
06: for _ in range(5):
07:     altitude, press = tphg.altitude(1029.1)
08:     print("Altitude = %d m, Pressure = %d"%(altitude, press))
09:     time.sleep(1)
```

```
PS C:\XNode\CORE> xnode -p com3 run core_tphg_altitude.py
Altitude = 90 m, Pressure = 1018 hPa
Altitude = 90 m, Pressure = 1018 hPa
Altitude = 90 m, Pressure = 1018 hPa
Altitude = 90 m, Pressure = 1018 hPa
Altitude = 90 m, Pressure = 1018 hPa
```

주변 환경 센싱 예제 5

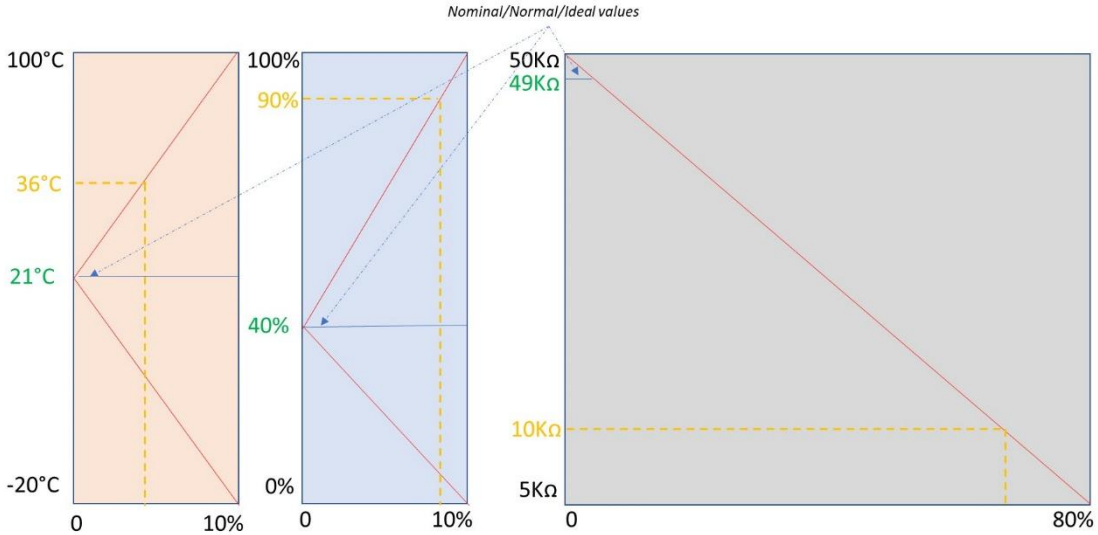
- 공기 질
  - 사용자에 따라 상대적이지만 몇 가지 연구가 있음

Rating	Excellent (5 points each)	Good (4 points each)	Fair (3 points each)	Poor (2 points each)	Inadequate (1 point each)
Temperature (°C)	18-21°C	Plus or minus 1°C *(including variance in occupied rooms, seasons and times of day)*	Plus or minus 2°C *(including variance in occupied rooms, seasons and times of day)*	Plus or minus 3°C *(including variance in occupied rooms, seasons and times of day)*	Plus or minus 4°C *or more (including variance in occupied rooms, seasons and times of day)*
Carbon Dioxide (PPM)	< 600	601 - 800	801 - 1500	1501 - 1800	> 1801
Relative Humidity (% RH)	40 - 60	< 40 / > 60	< 30 / > 70	< 20 / > 80	< 10 / > 90
Carbon Monoxide mg/m^3	0	-	1 - 7	-	7 >
Nitrogen Dioxide (mg/m^3)	< 0.2	-	0.2 – 0.4	-	0.4 >
TVOC (mg/m^3)	< 0.1	0.1 – 0.3	0.3 – 0.5	0.5 – 1.0	1.0 >

* 출처: IAQ UK

주변 환경 센싱 예제 5

- 온도, 습도, 가스 저항값에 따른 공기 질 관계
  - 온도 21도, 습도 40%, 가스 저항 50kohm일 때 가장 깨끗
    - 온도 36도, 습도 90%, 가스 저항 5kohm이면 불량



* 출처: Copyright (c) David Bird 2018

주변 환경 센싱 예제 5

- 습도(25% 가중치)와 가스(75% 가중치) 측정값으로 공기 질 분류
  - 가스 참조 값은 일정 시간마다 업데이트

```
01: from pop import Tphg
02: import time
03:
04: humi_ref = 40
05: gas_ref = 2500
06: gas_lower_limit = 10000 #bad ari quality limit
07: gas_upper_limit = 280000 #good air quality limit
08:
09: tphg = Tphg()
10:
11: for _ in range(10):
12:     _, _, _, g = tphg.read()
13:     gas_ref += g
14: gas_ref = gas_ref / 10
15:
```

주변 환경 센싱 예제 5

- (계속)

```
16: for i in range(1, 100+1):
17:     if not i % 5:
18:         for _ in range(10):
19:             _, _, _, g = tphg.read()
20:             gas_ref += g
21:         gas_ref = gas_ref / 10
22:
23:     _, _, humi, gas = tphg.read()
24:     print("Humi = %.2f RH, Gas = %d ohm"%(humi, gas), end=", ")
25:
26:     if humi >= 38 and humi <= 42:
27:         humi_score = 0.25 * 100
28:     else:
29:         if humi < 38: humi_score = (0.25 / humi_ref) * humi * 100
30:         else: humi_score = ((-0.25 / (100 - humi_ref) * humi) + 0.416666) * 100
31:
32:     gas_score = (0.75 / (gas_upper_limit - gas_lower_limit) * gas_ref - (gas_lower_limit * (0.75 /
33:                     (gas_upper_limit - gas_lower_limit)))) * 100.00
```

주변 환경 센싱 예제 5

- (계속)

```
34:    if gas_score > 75: gas_score = 75
35:    elif gas_score < 0: gas_score = 0
36:
37:    air_quality = (100 - (humi_score + gas_score)) * 5
38:    if air_quality > 300:
39:        txt = "Inadequate"
40:    elif air_quality > 200:
41:        txt = "Poor"
42:    elif air_quality > 100:
43:        txt = "Fair"
44:    elif air_quality > 50:
45:        txt = "Good"
46:    else:
47:        txt = "Excellent"
48:    print("Air = %.1f, %s"%(air_quality, txt))
```

주변 환경 센싱 예제 5

- 공기 질 분류는 일정 시간이 지나야 안정화됨
  - 환경에 따라 gas_lower_limit, gas_upper_limit 값을 조정해 보정

```
PS C:\XNode\CORE> xnode -p com3 run core_tphg_iaq.py
Humi = 25.38 RH, Gas = 217422 ohm, Air = 170.2, Fair
Humi = 25.41 RH, Gas = 218795 ohm, Air = 170.1, Fair
Humi = 25.42 RH, Gas = 221451 ohm, Air = 170.1, Fair
Humi = 25.42 RH, Gas = 222304 ohm, Air = 170.1, Fair
Humi = 25.46 RH, Gas = 229518 ohm, Air = 93.8, Good
Humi = 25.44 RH, Gas = 227260 ohm, Air = 93.9, Good
Humi = 25.43 RH, Gas = 228760 ohm, Air = 93.9, Good
Humi = 25.43 RH, Gas = 229366 ohm, Air = 93.9, Good
Humi = 25.42 RH, Gas = 228610 ohm, Air = 93.9, Good
Humi = 25.36 RH, Gas = 229975 ohm, Air = 81.7, Good
```

주변 환경 센싱 예제 5

- 재사용성을 높이기 위해 공기 질 분류 기능을 클래스로 구현

```
01: from pop import Tphg
02: import time
03:
04: class AirQuality(Tphg):
05:     def __init__(self):
06:         super().__init__()
07:         self.humi_ref = 40
08:         self.gas_ref = 2500
09:         self.gas_lower_limit = 10000 #bad ari quality limit
10:         self.gas_upper_limit = 280000 #good air quality limit
11:         self.count = 0
12:
13:         self._get_gas_ref()
14:
```

주변 환경 센싱 예제 5

- (계속)

```
15:     def _get_gas_ref(self):
16:         for _ in range(10):
17:             _, _, _, g = super().read()
18:             self.gas_ref += g
19:         self.gas_ref = self.gas_ref / 10
20:
21:     def _get_humi_score(self, humi):
22:         if humi >= 38 and humi <= 42:
23:             humi_score = 0.25 * 100
24:         else:
25:             if humi < 38: humi_score = (0.25 / self.humi_ref) * humi * 100
26:             else: humi_score = ((-0.25 / (100 - self.humi_ref) * humi) + 0.416666) * 100
27:
28:         return humi_score
29:
```

주변 환경 센싱 예제 5

- (계속)

```
30:     def _get_gas_score(self, gas):
31:         gas_score = (0.75 / (self.gas_upper_limit - self.gas_lower_limit) * self.gas_ref - (self.gas_lower_limit * (0.75 /
32:                         (self.gas_upper_limit - self.gas_lower_limit)))) * 100.00
33:         if gas_score > 75: gas_score = 75
34:         elif gas_score < 0: gas_score = 0
35:
36:         return gas_score
37:
38:     def read(self):
39:         self.count += 1
40:         if self.count % 5: self._get_gas_ref()
41:         _, _, humi, gas = super().read()
42:
43:         air_quality = (100 - (self._get_humi_score(humi) + self._get_gas_score(gas))) * 5
44:         if air_quality > 300:
45:             txt = "Inadequate"
```

주변 환경 센싱 예제 5

- (계속)

```
46:         elif air_quality > 200:
47:             txt = "Poor"
48:         elif air_quality > 100:
49:             txt = "Fair"
50:         elif air_quality > 50:
51:             txt = "Good"
52:         else:
53:             txt = "Excellent"
54:         return air_quality, txt, humi, gas
55:
56: #Test Code
57: aq = AirQuality()
58: for i in range(1, 100+1):
59:     air_quality, txt, humi, gas = aq.read()
60:     print("Humi = %.2f RH, Gas = %d ohm, Air = %.1f, %s"%(humi, gas, air_quality, txt))
```

```
PS C:\XNode\CORE> xnode -p com3 run core_tphg_iaq_class.py
Humi = 30.69 RH, Gas = 148860 ohm, Air = 237.2, Poor
Humi = 29.88 RH, Gas = 179481 ohm, Air = 173.5, Fair
Humi = 29.16 RH, Gas = 205071 ohm, Air = 127.1, Fair
Humi = 28.55 RH, Gas = 219071 ohm, Air = 100.0, Good
Humi = 28.50 RH, Gas = 218795 ohm, Air = 100.1, Fair
Humi = 28.00 RH, Gas = 228459 ohm, Air = 82.4, Good
Humi = 27.61 RH, Gas = 235124 ohm, Air = 70.7, Good
```

주변 환경 센싱 예제 6

- machine의 I2C 클래스로 환경 센싱
  - Pop의 Tphg 클래스는 machine의 I2C 클래스로 구현됨
  - machine 라이브러리의 I2C 클래스와 time, struct 모듈 로드
    - from machine import I2C
    - import time, struct
  - 환경 센서가 연결된 I2C 버스 1을 인자로 I2C 객체 생성
    - i2c = I2C (1)
  - 초기화 절차 진행
    - 리셋
      - i2c.writeto_mem(0x77, 0xE0, bytes([0xB6]))
      - time.sleep(0.01)

주변 환경 센싱 예제 6

- 환경 센서로부터 보정 데이터 읽기
  - calibration = i2c.readfrom_mem(0x77, 0x89, 25)
  - calibration += i2c.readfrom_mem(0x77, 0xE1, 16)
  - calibration = list(struct.unpack("<hbBHhbBhhbbHhhBBBHbbbBbHhbb", bytes(calibration[1:39])))
  - calibration = [float(i) for i in calibration]

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38
0		1	2	3		4		5	6	7		8		9	10	11		12		13		14	15	16	17		18	19	20	21	22	23		24		25	26
T2		T3		P1		P2		P3		P4		P5		P7	P6			P8		P9		P10		H2	H2/H1	H1	H3	H4	H5	H6	H7	T1		G2		G1	G3
						L	M			L	M	L	M					L	M	L	M			M	L	M						L	M	L	M		
h		b	B	H		h		b	B	h		h		b	b	H		h		h		B	B	B	H		b	b	b	B	b	H		h		b	b

주변 환경 센싱 예제 6

- 읽은 보정 데이터에서 온도, 기압, 습도 보정 계수 분리 및 가스 보정 계수 읽기
  - temp_calibration = [calibration[x] for x in [23, 0, 1]]
  - pressure_calibration = [calibration[x] for x in [3, 4, 5, 7, 8, 10, 9, 12, 13, 14]]
  - humidity_calibration = [calibration[x] for x in [17, 16, 18, 19, 20, 21, 22]]
  - humidity_calibration[0] /= 16 #습도 상, 하 바이트 치환
  - humidity_calibration[1] *= 16
  - humidity_calibration[1] += humidity_calibration[0] % 16
  - sw_err = (i2c.readfrom_mem(0x77, 0x04, 1)[0] & 0xF0) / 16
- 히터 목표 저항(115) 및 히터가 데워질 때까지의 대기시간(100ms) 설정
  - i2c.writeto_mem(0x77, 0x5A, bytes([0x73]))
  - i2c.writeto_mem(0x77, 0x64, bytes([0x65]))

주변 환경 센싱 예제 6

- 측정 및 결과 얻기
  - 디지털 필터 계수(3) 선택 및 온도(x5), 기압(x4), 습도(x2) 최대 샘플링 수 설정
    - i2c.writeto_mem(0x77, 0x75, bytes([0b010 << 2]))
    - i2c.writeto_mem(0x77, 0x74, bytes([(0b100 << 5) | (0b011 << 2)]))
    - i2c.writeto_mem(0x77, 0x72, bytes([0b010]))
  - 사용할 히터 설정 인덱스(0) 및 가스 측정 시작 설정
    - i2c.writeto_mem(0x77, 0x71, bytes([0x10]))
  - 운영 모드를 포스 모드(0b01)로 변경해 측정 시작
    - ctrl = i2c.readfrom_mem(0x77, 0x74, 1)[0]
    - ctrl = (ctrl & 0xFC) | 0x01
    - i2c.writeto_mem(0x77, 0x74, bytes([ctrl]))

주변 환경 센싱 예제 6

- 내장 ADC에서 측정 데이터 읽기
  - new_data = False
  - while not new_data:

    data = i2c.readfrom_mem(0x77, 0x1D, 15)

    new_data = data[0] & 0x80 != 0

    time.sleep(0.005)
- 온도, 기압, 습도, 가스 데이터 분리
  - adc_pres = ((data[2] * 4096) + (data[3] * 16) + (data[4] / 16))
  - adc_temp = ((data[5] * 4096) + (data[6] * 16) + (data[7] / 16))
  - adc_hum = struct.unpack(">H", bytes(data[8:10]))[0]
  - adc_gas = int(struct.unpack(">H", bytes(data[13:15]))[0] / 64)
  - gas_range = data[14] & 0x0F

주변 환경 센싱 예제 6

- 기준 온도 계산
  - var1 = (adc_temp / 8) - (temp_calibration[0] * 2)
  - var2 = (var1 * temp_calibration[1]) / 2048
  - var3 = ((var1 / 2) * (var1 / 2)) / 4096
  - var3 = (var3 * temp_calibration[2] * 16) / 16384
  - t_fine = int(var2 + var3)
- 기준 온도로부터 섭씨온도 계산
  - calc_temp = ((t_fine * 5) + 128) / 256
  - print("Temp = %.2f"%(calc_temp / 100))

주변 환경 센싱 예제 6

- 기준 온도를 이용해 기압 계산
  - var1 = (t_fine / 2) - 64000
  - var2 = ((var1 / 4) * (var1 / 4)) / 2048
  - var2 = (var2 * pressure_calibration[5]) / 4
  - var2 = var2 + (var1 * pressure_calibration[4] * 2)
  - var2 = (var2 / 4) + (pressure_calibration[3] * 65536)
  - var1 = ((((var1 / 4) * (var1 / 4)) / 8192) * (pressure_calibration[2] * 32) / 8) + ((pressure_calibration[1] * var1) / 2)
  - var1 = var1 / 262144
  - var1 = ((32768 + var1) * pressure_calibration[0]) / 32768
  - calc_pres = 1048576 - adc_pres
  - calc_pres = (calc_pres - (var2 / 4096)) * 3125
  - calc_pres = (calc_pres / var1) * 2
  - var1 = (pressure_calibration[8] * (((calc_pres / 8) * (calc_pres / 8)) / 8192)) / 4096
  - var2 = ((calc_pres / 4) * pressure_calibration[7]) / 8192
  - var3 = (((calc_pres / 256) ** 3) * pressure_calibration[9]) / 131072
  - calc_pres += (var1 + var2 + var3 + (pressure_calibration[6] * 128)) / 16
  - print("Press = %.2f"%(calc_pres / 100))

주변 환경 센싱 예제 6

- 기준 온도를 이용해 상대 습도 계산
  - temp_scaled = ((t_fine * 5) + 128) / 256
  - var1 = (adc_hum - (humidity_calibration[0] * 16)) - ((temp_scaled * humidity_calibration[2]) / 200)
  - var2 = (humidity_calibration[1] * (((temp_scaled * humidity_calibration[3]) / 100) +
    (((temp_scaled * ((temp_scaled * humidity_calibration[4]) / 100)) / 64) / 100) + 16384)) / 1024
  - var3 = var1 * var2
  - var4 = humidity_calibration[5] * 128
  - var4 = (var4 + ((temp_scaled * humidity_calibration[6]) / 100)) / 16
  - var5 = ((var3 / 16384) * (var3 / 16384)) / 1024
  - var6 = (var4 * var5) / 2
  - calc_hum = (((var3 + var6) / 1024) * 1000) / 4096
  - calc_hum /= 1000
  - print("Humi = %.2f"%(100 if calc_hum > 100 else 0 if calc_hum < 0 else calc_hum))

주변 환경 센싱 예제 6

- 유해 가스 감지 계산
  - LOOKUP_TABLE_1 = (2147483647.0, 2147483647.0, 2147483647.0, 2147483647.0, 2147483647.0, 2126008810.0, 2147483647.0, 2130303777.0, 2147483647.0, 2147483647.0, 2143188679.0, 2136746228.0, 2147483647.0, 2126008810.0, 2147483647.0, 2147483647.0)
  - LOOKUP_TABLE_2 = (4096000000.0, 2048000000.0, 1024000000.0, 512000000.0, 255744255.0, 127110228.0, 64000000.0, 32258064.0, 16016016.0, 8000000.0, 4000000.0, 2000000.0, 1000000.0, 500000.0, 250000.0, 125000.0)
  - var1 = ((1340 + (5 * sw_err)) * (LOOKUP_TABLE_1[gas_range])) / 65536
  - var2 = ((adc_gas * 32768) - 16777216) + var1
  - var3 = (LOOKUP_TABLE_2[gas_range] * var1) / 512
  - print("Gas = %d"%((var3 + (var2 / 2)) // var2)

주변 환경 센싱 예제 6

- 저수준 BME 680 환경 센싱 구현

```
1.  from machine import I2C
2.  import time, struct
3.
4.  i2c = I2C (1)
5.
6.  i2c.writeto_mem(0x77, 0xE0, bytes([0xB6]))
7.  time.sleep(0.01)
8.
9.  calibration = i2c.readfrom_mem(0x77, 0x89, 25)
10. calibration += i2c.readfrom_mem(0x77, 0xE1, 16)
11. calibration = list(struct.unpack("<hbBHhbBhhbbHhhBBBHbbbBbHhbb", bytes(calibration[1:39])))
12. calibration = [float(i) for i in calibration]
13.
14. temp_calibration = [calibration[x] for x in [23, 0, 1]]
15. pressure_calibration = [calibration[x] for x in [3, 4, 5, 7, 8, 10, 9, 12, 13, 14]]
16. humidity_calibration = [calibration[x] for x in [17, 16, 18, 19, 20, 21, 22]]
17. humidity_calibration[0] /= 16
18. humidity_calibration[1] *= 16
19. humidity_calibration[1] += humidity_calibration[0] % 16
20. sw_err = (i2c.readfrom_mem(0x77, 0x04, 1)[0] & 0xF0) / 16
```

환경 센싱 예제 6

- (계속)

```
21. i2c.writeto_mem(0x77, 0x5A, bytes([0x73]))
22. i2c.writeto_mem(0x77, 0x64, bytes([0x65]))
23.
24. i2c.writeto_mem(0x77, 0x75, bytes([0b010 << 2]))
25. i2c.writeto_mem(0x77, 0x74, bytes([(0b100 << 5) | (0b011 << 2)]))
26. i2c.writeto_mem(0x77, 0x72, bytes([0b010]))
27.
28. i2c.writeto_mem(0x77, 0x71, bytes([0x10]))
29.
30. for _ i n range(10):
31.     ctrl = i2c.readfrom_mem(0x77, 0x74, 1)[0]
32.     ctrl = (ctrl & 0xFC) | 0x01
33.     i2c.writeto_mem(0x77, 0x74, bytes([ctrl]))
34.
35.     new_data = False
36.     while not new_data:
37.         data = i2c.readfrom_mem(0x77, 0x1D, 15)
38.         new_data = data[0] & 0x80 != 0
39.         time.sleep(0.005)
```

환경 센싱 예제 6

- (계속)

```
40.   adc_pres = ((data[2] * 4096) + (data[3] * 16) + (data[4] / 16))
41.   adc_temp = ((data[5] * 4096) + (data[6] * 16) + (data[7] / 16))
42.   adc_hum = struct.unpack(">H", bytes(data[8:10]))[0]
43.   adc_gas = int(struct.unpack(">H", bytes(data[13:15]))[0] / 64)
44.   gas_range = data[14] & 0x0F
45.
46.   var1 = (adc_temp / 8) - (temp_calibration[0] * 2)
47.   var2 = (var1 * temp_calibration[1]) / 2048
48.   var3 = ((var1 / 2) * (var1 / 2)) / 4096
49.   var3 = (var3 * temp_calibration[2] * 16) / 16384
50.   t_fine = int(var2 + var3)
51.
52.   calc_temp = ((t_fine * 5) + 128) / 256
53.   print("Temp = %.2f"%(calc_temp / 100), end=", ")
54.
55.   var1 = (t_fine / 2) - 64000
56.   var2 = ((var1 / 4) * (var1 / 4)) / 2048
57.   var2 = (var2 * pressure_calibration[5]) / 4
58.   var2 = var2 + (var1 * pressure_calibration[4] * 2)
```

환경 센싱 예제 6

- (계속)

```
59.    var2 = (var2 / 4) + (pressure_calibration[3] * 65536)
60.    var1 = ((((var1 / 4) * (var1 / 4)) / 8192) * (pressure_calibration[2] * 32) / 8) + ((pressure_calibration[1] * var1) / 2)
61.    var1 = var1 / 262144
62.    var1 = ((32768 + var1) * pressure_calibration[0]) / 32768
63.    calc_pres = 1048576 - adc_pres
64.    calc_pres = (calc_pres - (var2 / 4096)) * 3125
65.    calc_pres = (calc_pres / var1) * 2
66.    var1 = (pressure_calibration[8] * (((calc_pres / 8) * (calc_pres / 8)) / 8192)) / 4096
67.    var2 = ((calc_pres / 4) * pressure_calibration[7]) / 8192
68.    var3 = (((calc_pres / 256) ** 3) * pressure_calibration[9]) / 131072
69.    calc_pres += (var1 + var2 + var3 + (pressure_calibration[6] * 128)) / 16
70.    print("Press = %.2f"%(calc_pres / 100), end=", ")
71.
72.    temp_scaled = ((t_fine * 5) + 128) / 256
73.    var1 = (adc_hum - (humidity_calibration[0] * 16)) - ((temp_scaled * humidity_calibration[2]) / 200)
74.    var2 = (humidity_calibration[1] * (((temp_scaled * humidity_calibration[3]) / 100) +
75.            (((temp_scaled * ((temp_scaled * humidity_calibration[4]) / 100)) / 64) / 100) + 16384)) / 1024
76.    var3 = var1 * var2
77.    var4 = humidity_calibration[5] * 128
```

환경 센싱 예제 6

- (계속)

```
78.   var4 = (var4 + ((temp_scaled * humidity_calibration[6]) / 100)) / 16
79.   var5 = ((var3 / 16384) * (var3 / 16384)) / 1024
80.   var6 = (var4 * var5) / 2
81.   calc_hum = (((var3 + var6) / 1024) * 1000) / 4096
82.   calc_hum /= 1000
83.   print("Humi = %.2f"%(100 if calc_hum > 100 else 0 if calc_hum < 0 else calc_hum), end=", ")
84.
85.   LOOKUP_TABLE_1 = (2147483647.0, 2147483647.0, 2147483647.0, 2147483647.0, 2147483647.0,
86.             2126008810.0, 2147483647.0, 2130303777.0, 2147483647.0, 2147483647.0, 2143188679.0,
87.             2136746228.0, 2147483647.0, 2126008810.0, 2147483647.0, 2147483647.0)
88.   LOOKUP_TABLE_2 = ( 4096000000.0, 2048000000.0, 1024000000.0, 512000000.0, 255744255.0,
89.             127110228.0, 64000000.0, 32258064.0, 16016016.0, 8000000.0, 4000000.0, 2000000.0,
90.             1000000.0, 500000.0, 250000.0, 125000.0)
91.   var1 = ((1340 + (5 * sw_err)) * (LOOKUP_TABLE_1[gas_range])) / 65536
92.   var2 = ((adc_gas * 32768) - 16777216) + var1
93.   var3 = (LOOKUP_TABLE_2[gas_range] * var1) / 512
94.   print("Gas = %d"%((var3 + (var2 / 2)) // var2))
95.
96.   time.sleep(1)
```

```
PS C:\XNode\CORE> xnode -p com3 run core_tphg_raw.py
Temp = 26.58, Press = 1015.14, Humi = 24.66, Gas = 284080
Temp = 26.47, Press = 1015.14, Humi = 24.84, Gas = 135786
Temp = 26.35, Press = 1015.13, Humi = 25.12, Gas = 139328
```

기본 센서 종합 예제 1

- 시리얼로 XNode 동작 제어
  - 한 문자 단위로 명령 정의
    - 'q': 종료, 'b': 배터리 전압, L: 조도 값, T: 환경(온도, 기압, 습도, 가스) 센서값

```
01: from pop import Battery, Light, Tphg
02: from pop import time
03: from pop import Uart
04:
05: battery = Battery()
06: light = Light()
07: tphg = Tphg()
08:
09: uart = Uart()
10:
11: uart.write("Start...\n")
12:
```

기본 센서 종합 예제 1

- (계속)

```
13: while True:
14:     cmd = uart.read(1).decode()
15:     if cmd == 'q':
16:         break
17:     elif cmd == 'b':
18:         uart.write("\nBattery: %.2f Volt\n"%(battery.read()))
19:     elif cmd == 'L':
20:         uart.write("\nLight: %d lx\n"%(light.read()))
21:     elif cmd == 'T':
22:         uart.write("\nTemp: %.2f degree, Press: %.2f hPa, Humi: %.2f RH, Gas: %d ohm\n"%(tphg.read()))
23:     else:
24:         uart.write("\nKnown command\n")
25:
26: uart.write("\nThe End...")
```

기본 센서 종합 예제 1

- 입력한 문자 확인을 위해 -i 옵션으로 프로그램 실행
  - 문자를 입력할 때마다 결과 반환

```
PS C:\XNode\CORE> xnode -p com3 run -i core_total_uart.py
Start...
b
Battery: 3.80 Volt
L
Light: 341 lx
T
Temp: 13.94 degree, Press: 1018.25 hPa, Humi: 19.36 RH, Gas: 67146 ohm
x
Known command
q
The End...
```

기본 센서 종합 예제 2

- 오프라인 처리를 위해 센서 데이터를 XNode의 /flash 경로에 파일로 저장
  - 일정 기간 동안 또는 이동하면서 모은 데이터를 파일로 보관한 후 필요할 때 분석
    - XNode에 저장된 파일을 PC로 옮긴 후 사용
  - Pop 라이브러리 설치 공간을 고려할 때 남은 공간은 약 300KByte
    - 사용자 프로그램을 설치하면 공간은 더 줄어듦
    - XNode에 300KByte 크기의 파일이 있을 때 이를 PC로 옮기는 데 약 30초 소요
      - xnode -p 포트 get <XNode 파일명> <PC 저장 파일명>

기본 센서 종합 예제 2

- 내장 함수 open()으로 파일 객체를 만든 후 읽기 또는 쓰기 진행
  - open(file, mode): 파일 객체를 생성한 후 반환
    - file: 파일 이름
    - mode: 'r' (읽기:기본값), 'w' (쓰기) 중 하나
      - XNode는 현재 'a'(추가)를 지원하지 않으므로 기존 파일을 열어 내용을 추가할 수 없음
    - 반환은 파일 객체
  - *file_object*.write(data): 파일 쓰기
    - data: 문자열. 읽을 때 줄 단위로 읽으려면 끝에 줄 바꿈 문자('₩n')를 포함해야 함
  - *file_object*.readline(): 줄 단위 파일 읽기
    - 반환은 줄 단위 문자열. 더 이상 읽을 줄이 없으면 None

기본 센서 종합 예제 2

- os 모듈은 파일 시스템 수정 기능 제공
  - 루트 디렉토리는 /flash임
  - os.listdir([dir]): 해당 디렉터리의 파일 목록 반환
    - dir: 디렉터리 이름. 생략하면 현재 디렉터리
    - 파일 목록을 리스트로 반환
  - os.mkdir(dir): 새로운 디렉터리 생성
    - dir: 디렉터리 이름
  - os.rmdir(dir): 디렉터리 제거
    - dir: 디렉터리 이름
  - os.remove(file): 파일 제거
    - file: 파일 이름

기본 센서 종합 예제 2

- 1초 단위로 Pop을 이용해 LED 토글 및 배터리 잔량, 조도, 환경 센서 측정
  - 측정 결과는 "배터리 잔량, 조도, 온도, 기압, 습도, 가스" 순으로 sensors.dat 파일에 저장
    - 지속적인 테스트를 위해 XNode에 sensors.dat 파일이 존재하면 삭제한 후 다시 만듦

```
01: from pop import Battery, Light, Tphg
02: from pop import time
03: import os
04:
05: battery = Battery()
06: light = Light()
07: tphg = Tphg()
08:
09: if "sensors.dat" in os.listdir():
10:     os.remove("sensors.dat")
11:
```

기본 센서 종합 예제 2

- (계속)

```
12: f = open("sensors.dat", "w")
13: f.write("BATTERY, LIGHT, TEMP, PRESS, HUMI, GAS\n")
14:
15: for _ in range(10):
16:
17:     v = battery.read()
18:     l = light.read()
19:     t, p, h, g = tphg.read()
20:
21:     data = "%.2f,%d,%.2f,%.2f,%.2f,%d"%(v, l, t, p, h, g)
22:     print(data)
23:     f.write(data+'\n')
24:
25:     time.sleep(1)
26:
27: f.close()
```

기본 센서 종합 예제 2

- 프로그램을 실행하면 센서 측정 결과를 화면에 출력하면서 파일로 저장
  - 현재 XNode는 파일 내용 추가 모드를 지원하지 않으므로 기존 파일을 재사용해 저장할 수 없음

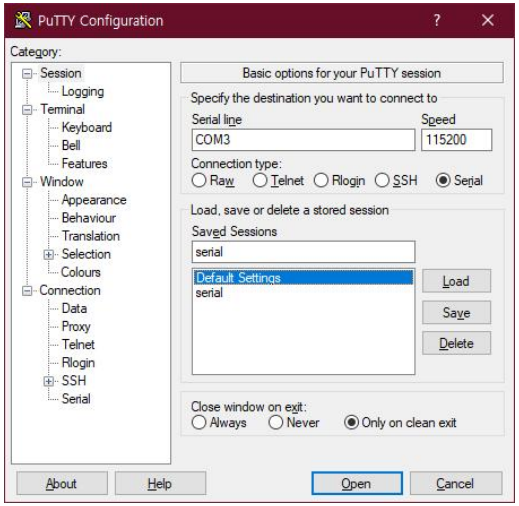
```
PS C:\XNode\CORE> xnode -p com3 run core_total_file_write.py
0.00,87,23.73,1023.74,15.69,17172
0.00,261,23.75,1023.74,15.68,6590
0.00,147,23.79,1023.73,15.67,7422
0.00,0,23.86,1023.74,15.66,7965
0.00,1,24.37,1023.77,16.15,8246
0.00,2,24.90,1023.77,17.65,8689
0.00,197,25.25,1023.77,19.48,9427
0.00,117,25.37,1023.77,20.59,10302
0.00,242,25.60,1023.77,20.99,10734
0.00,265,25.65,1023.78,21.29,12032
```

기본 센서 종합 예제 2

- XNode 재부팅 후 프로그램 자동 실행
  - XNode는 재부팅될 때 /flash/main.py를 찾고, 존재하면 이를 실행함.
  - 작성한 프로그램을 XNode에 복사할 때 main.py로 이름을 바꿈

```
PS C:\XNode\CORE> xnode -p com3 put core_total_file_write.py main.py
```

- 출력 결과를 확인하려면 시리얼 프로그램(PuTTY 등) 실행
  - 현재 XNode 포트와 전송 속도를 115200으로 설정한 후 실행
  - 이후 XNode의 리셋 버튼을 누름



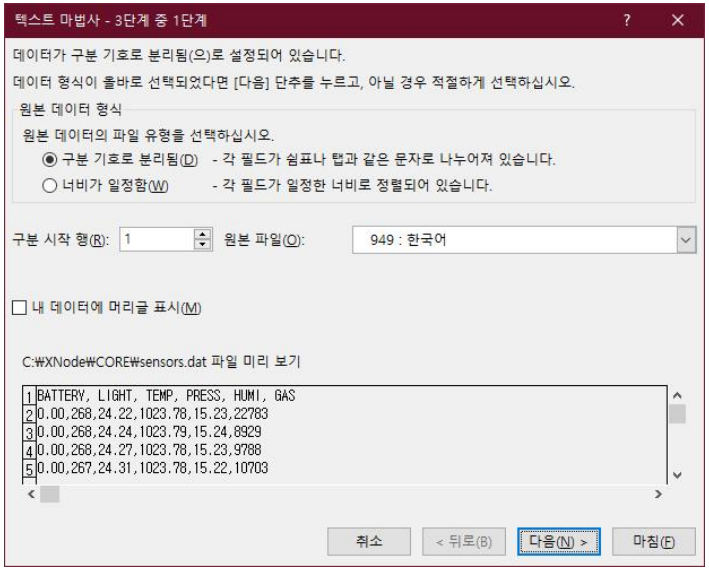
기본 센서 종합 예제 2

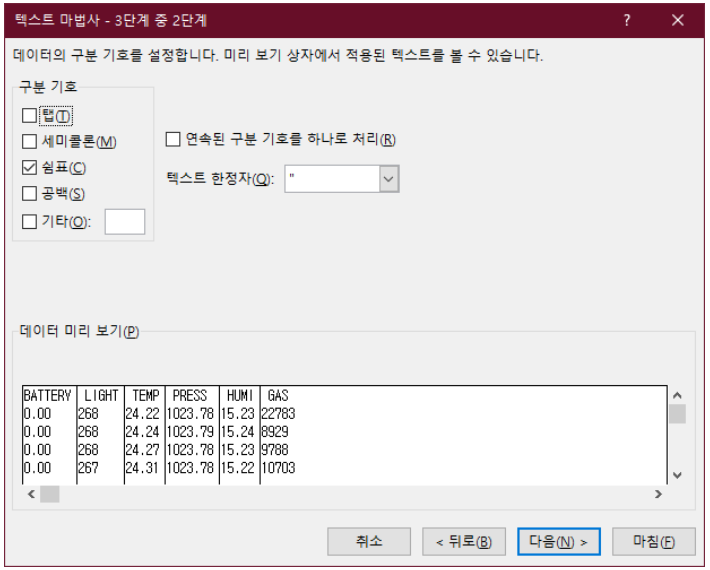
- 저장된 파일은 PC로 옮겨 분석
  - 의미 있는 정보를 얻으면 실행 시간이 길어야(주 단위) 함.

```
PS C:\XNode\CORE> xnode -p com3 get sensors.dat sensors.dat
PS C:\XNode\CORE> dir sensors.dat


    디렉터리: C:\XNode\CORE


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----        2020-12-01   오후 3:39            387 sensors.dat
```

기본 센서 종합 예제 2

- Excel 실행 후 PC에 저장한 sensors.dat 파일 열기

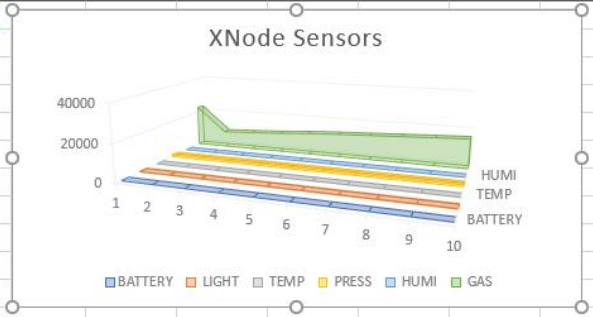




기본 센서 종합 예제 2

- Excel 시트에서 데이터 항목을 선택한 후 차트 그리기

	A	B	C	D	E	F
1	BATTERY	LIGHT	TEMP	PRESS	HUMI	GAS
2	3.8	268	24.22	1023.78	15.23	22783
3	3.8	268	24.24	1023.79	15.24	8929
4	3.8	268	24.27	1023.78	15.23	9788
5	3.8	267	24.31	1023.78	15.22	10703
6	3.8	268	24.34	1023.76	15.22	12013
7	3.8	268	24.37	1023.76	15.19	12967
8	3.8	268	24.4	1023.76	15.19	13804
9	3.8	268	24.43	1023.76	15.17	14629
10	3.8	268	24.45	1023.76	15.16	15450
11	3.8	268	24.47	1023.76	15.13	16223
12						
13						



기본 센서 종합 예제 3

- XNode에서 측정 결과가 저장된 sensors.dat 파일 내용을 읽어 처리하는 것도 가능
  - sensors.dat 파일 내용 출력
  - Sensors.dat 파일은 앞의 예제에 Xnode의 /flash 위치에 생성됨

```
01: from pop import time
02:
03: f = open("sensors.dat")
04:
05: while True:
06:     data = f.readline()
07:     if not data:
08:         break
09:     print(data,end='')
10:
11: f.close()
```

```
PS C:\XNode\CORE> xnode -p com3 run core_total_file_read.py
BATTERY, LIGHT, TEMP, PRESS, HUMI, GAS
0.00,87,23.73,1023.74,15.69,17172
0.00,261,23.75,1023.74,15.68,6590
0.00,147,23.79,1023.73,15.67,7422
0.00,0,23.86,1023.74,15.66,7965
0.00,1,24.37,1023.77,16.15,8246
0.00,2,24.90,1023.77,17.65,8689
0.00,197,25.25,1023.77,19.48,9427
0.00,117,25.37,1023.77,20.59,10302
0.00,242,25.60,1023.77,20.99,10734
0.00,265,25.65,1023.78,21.29,12032
```