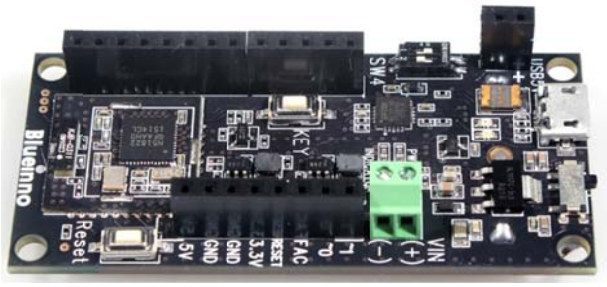
블루이노2 함수 사용방법

Ver = 02

No.	Date	내용
1	2015년 6월 15일	analogReference, analogSelection 함수 추가
2	2015년 10월 09일	문서 자료 보강 Sensor beacon 기능 추가
3	2015년 11월 28일	IO Current 함수 보강, Interrupt, SoftwareSerial
4	2016년 06월 03일	Device Name 변경 ; 비콘 방식에서 오류 수정



Blueinno
Bluetooth Innovation

블루이노 (www.blueinno.co.kr)

우: 153-715

서울특별시 금천구 벚꽃로 298, 514호 (가산동, 대륭포스트타워 6차)

T : 070-4288-8187 F : 02-2083-8188 (dwlee@blueinno.co.kr)

[BLE Stack관련]

블루투스 4.0(BLE)의 기능을 사용하기 위한 방법으로 RFduinoBLE.h파일을 include시켜야 하며 이후 아래에 설명한 함수를 사용하게 됩니다.

- RFduinoBLE.begin()

이 함수는 BLE Stack의 시작함을 나타냅니다. BLE 기능이 시작되면 기본 세팅된 시간 간격으로 advertising(방송)을 시작하게 됩니다. 이를 통하여 폰과의 연결시 어떤 서비스를 하는것인지를 알수있습니다.

참고사항으로 Beacon 앱과 ble(페어링) 앱이 동시에 검색됩니다.

- Device Name 변경

1) Beacon 방식은 예제/BlueinnoBLE/AdvertisementRawiBeacon 를 실행하셔서 길이와 0x09, 이후에 해당 디바이스 이름을 입력하면

됩니다. 꼭! 길이를 29글자로 하셔야 변경이 됩니다.

==========

```
#include <RFduinoBLE.h>
uint8_t advdata[] ={
  30,     // length
  0x09,   // complete local name type
  0x41,   // 'A'
  0x42,   // 'B'
  0x43,   // 'C'
   ~~~~
  0x32,   // '3'
};
```

2) BLE(페어링) 앱은 **RFduinoBLE.cpp file내의** 저장된

- RFduinoBLE.deviceName = "blueinno; //device이름을 blueinno 로 표시.
- RFduinoBLE.advertismentData = "sketch"; //advertisement 패킷에 “sketch”가 포함

- RFduinoBLE.end()

이 함수는 BLE Stack의 정지함을 나타냅니다.

- RFduinoBLE.deviceName

이 변수는 BLE device이름을 advertising시에 보내어집니다. 폰에서는 본 데이터를 받아서 관련 디바이스 이름을 확인하여 서비스 시작 유무에 이용할수도 있습니다.

예: **RFduinoBLE.cpp file내의**

RFduinoBLE.deviceName = "BLUEINNO"; //device이름을 blueinno 로 설정.

- RFduinoBLE.advertismentData

이 변수는 BLE advertisement 데이터를 설정할 수 있습니다.

예: **RFduinoBLE.cpp file내의**

RFduinoBLE.advertismentData = "sketch"; //advertisement 패킷에 "sketch"가 포함

- RFduinoBLE.advertisementInterval

이 변수는 밀리초(msec) 단위로 BLE advertisement 간격을 설정할 수 있습니다.

예:

RFduinoBLE.advertisementInterval = 100; //100ms 로 간격을 설정.

// interval between advertisement tx ms (range is 20ms to 10.24s) - default 80ms

- RFduinoBLE.txPowerLevel

이 변수는 dBm 단위로 BLE 전송 전력(transmit power)을 설정할 수 있고 4dBm 격차로 -20 ~ +4 dBm 사이의 값을 선택할 수 있습니다. (예. -20, -16, -12, -8, -4, 0, +4)

예:

RFduinoBLE.txPowerLevel = +4; //전송 전력을 최대 +4dBm 으로 설정.

// Defaul는 +4 dBm으로 설정되어 있습니다.

- RFduinoBLE.send()

이 함수를 사용하면 BLE를 통해 데이터를 보낼 수 있습니다. RFduinoBLE.send(char data) or RFduinoBLE.send(const char *data, int len);

예:

RFduinoBLE.send (1); //숫자 1 을 보냄.

RFduinoBLE.send (myarray, 5); //myarray라는 문자 배열에서 myarray[0]~[4]까지 5개 보냄.

예를들어 myarray[] = "hello" 라고 정의하였으면 hello가 송신됨

- RFduinoBLE.sendByte()

이 함수를 사용하면 BLE를 통해 Byte형 데이터를 보낼 수 있습니다.

예:

```
uint8_t myByte = 50;
RFduinoBLE.sendByte (myByte); //myByte를 보냄.
```

- RFduinoBLE.sendInt()

이 함수를 사용하면 BLE를 통해 INT형 데이터를 보낼 수 있습니다.

예:

```
int myByte = 5000;
RFduinoBLE.sendInt (myByte); //int형 myByte를 보냄.
```

- RFduinoBLE.sendFloat()

이 함수를 사용하면 BLE를 통해 float형 데이터를 보낼 수 있습니다.

예:

```
float myNumber = 16.49;
RFduinoBLE.sendFloat (myNumber); //float형 myNumber를 보냄.
```

- RFduinoBLE.radioActive

이 함수를 사용하면 무선(radio)이 활성화되어 있는지 여부를 확인할 수 있습니다. 활성화된 무선은 모든 resource에서 우선하므로, 중요한 코드를 실행하기 위해 무선이 off될 때가지 기다려야 되는 타이밍이 중요한 application에서 매우 유용합니다.

예:

```
// 무선이 활성화되어 있는 동안은 기다림.
while (RFduinoBLE.radioActive) ;
// 타이밍이 중요한 코드는 여기에 넣음.
```

[IBeacon관련]

- IBeacon은 블루투스 4.0(BLE) 프로토콜 기반의 근거리 무선통신 장치, 5~10cm 단위의 구별이 가능할 정도로 정확성이 있는 애플에 의해 2013년 9월 공개되었습니다.
- IBeacon을 이용하여 실내 네비게이션 및 빅데이터 수집, 기타 매장 서비스등이 가능합니다.

- RFduinoBLE.iBeacon

iBeacon 서비스의 advertising을 enable함

예: RFduinoBLE.iBeacon = true; //Enable iBeacon advertising

RFduinoBLE.begin(); //Start BLE

예 : 사용자 UUID, Major, Minor (Major, Minor를 이용하여 서비스에 대한 정의를 할수 있습니다, 예를 들어 Major로 층을 나타내고, Minor를 가지고 호수를 나타낼수 있습니다. 전파세기

RFduinoBLE.iBeacon = true; // iBeacon사용

uint8_t uuid[16] = {0xE2, 0xC5, 0x6D, 0xB5, 0xDF, 0xFB, 0x48, 0xD2, 0xB0, 0x60, 0xD0, 0xF5, 0xA7, 0x10, 0x96, 0xE0}; //Custom iBeacon UUID

memcpy(RFduinoBLE.iBeaconUUID, uuid, sizeof(RFduinoBLE.iBeaconUUID));

RFduinoBLE.iBeaconMajor = 1234; // Major

RFduinoBLE.iBeaconMinor = 5678; // Minor

RFduinoBLE.iBeaconMeasuredPower = 0xC6; //2's complement iBeacon Power Measurement at 1 Meter (default is 0xC5 = -59dBm

RFduinoBLE.begin(); //Start BLE stack

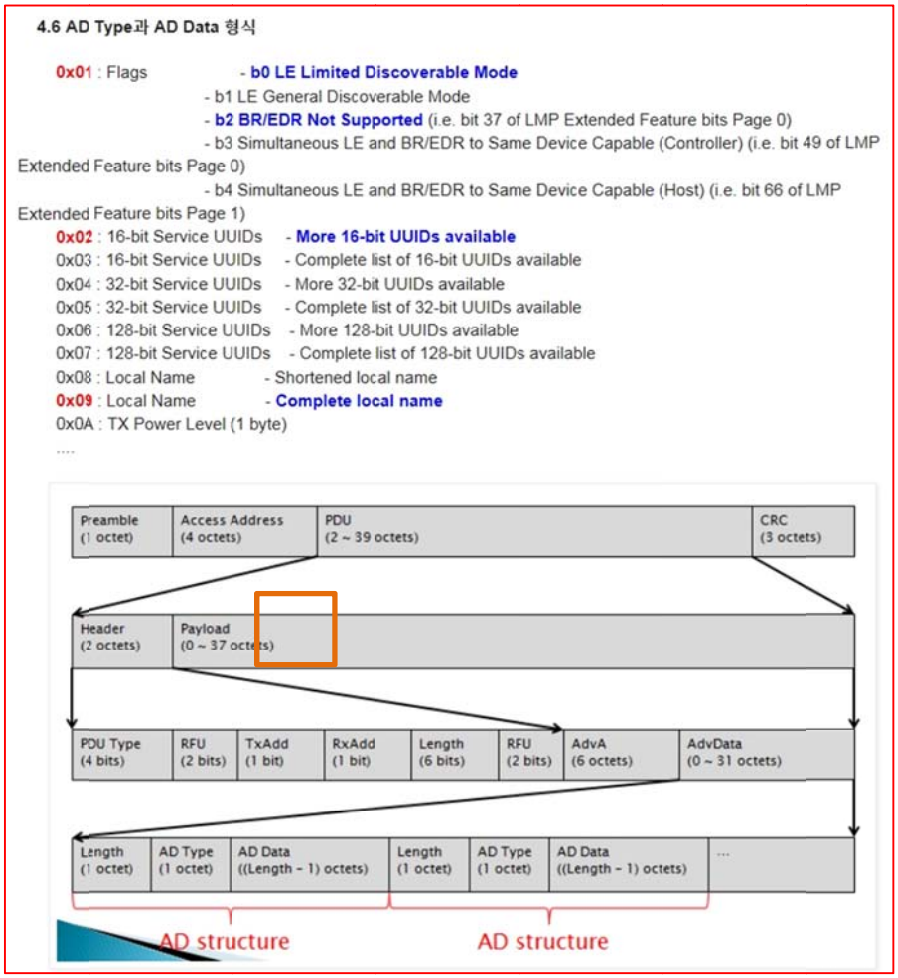
[Sensor Beacon]

```
#include <RFduinoBLE.h>

// the advertisement packet is composed of a series of variable length blocks, that can appear in any order.

// each block starts with a length byte, followed by a type byte, followed by the data.

// the payload cannot exceed 31 bytes.
```




```
uint8_t advdata[] =

{

0x02,   // length

0x01,   // flags type

0x04,   // br edr not supported
```

```
0x02,   // length

0x01,   // flags type

0x06,   // le general discovery mode | br edr
```

```
0x02,   // length

0x0A,   // tx power level

0x04,   // +4dBm
```

```
// if this variable block is not included, the RFduino iPhone apps won't see the device

0x03,   // length

0x03,   // 16 bit service uuid (complete)

0x20,   // uuid low

0x22,   // uuid hi
```

```
// Sensor Beacon
  0x0E, // length
  0x18, // type    sensor beacon
  0x00, // sensor ID(high)
  0x20, // sensor ID(low)
  0x00, // Service type 00-temp,humi, 01-??Sensor
  03,     // Voltage high
  01,     // Voltage low
  0x00, // Temp value float type , advdata[17]
  0x00,
  0x00,
  0x00,
  0x00, // Humi value float type
  0x00,
  0x00,
  0x00,
};
```

[BLE Callbacks]

- RFduinoBLE_onAdvertisement()

이 함수를 사용하면 advertising(주기적 송신, 연결되기 전 자신에 대한 정보를 계속적으로 송신함)이 되는 동안은 코드의 일부를 실행할 수 있습니다.

예:

```
void RFduinoBLE_onAdvertisement(bool start){
  if(start)
    digitalWrite(led, HIGH); // advertising하면 LED가 켜짐
  else
    digitalWrite(led, LOW); // advertising이 안하고 있으면 LED가 꺼짐
 }
```

- RFduinoBLE_onConnect()

무선이 폰과(기타 다른 host) 연결된 동안은 코드의 일부를 실행할 수 있습니다.

예:

```
void RFduinoBLE_onConnect(){
    digitalWrite(led, HIGH); // 폰과 연결이 되면 LED가 켜짐

 }
```

- RFduinoBLE_onDisconnect()

무선이 폰과(기타 다른 host) 연결되지 않는 동안은 코드의 일부를 실행할 수 있습니다.

예:

```
void RFduinoBLE_onDisconnect(){
  digitalWrite(led, HIGH); // 폰과 연결이 안되어져 있으면 LED가 켜짐
 }
```

- RFduinoBLE_onReceive()

블루이노가 폰(host)으로부터 데이터를 수신하였을 때 불리워지는 함수로 수신된 데이터를 표시하거나 필요한 동작을 할수 있습니다..

예:

```
void RFduinoBLE_onReceive(char *data, int len){
    uint8_t myByte = data[0]; // 배열에서 첫번째 char형 데이터를 myByte로 저장.
    Serial.println(myByte); // serial을 통해 myByte를 출력.
}
```

- RFduinoBLE_onRSSI()

연결후에 RSSI(수신 신호의세기)를 표시합니다.

예:

```
void RFduinoBLE_onRSSI(int rssi){
  Serial.println(rssi); // serial을 통해 rssi 값을 출력.
}
```

[GZLL Library]

Gazell(GZLL)은 블루투스 4.0 스팩이 아니며 독창적으로 1대(호스트) 8대(디바이스)가 연결될수 있도록 만든것으로 1개의 호스트가 8개의 디바이스와 연결되어서 필요한 정보를 수신 혹은 송신을 할수 있는 프로토콜입니다.

- device_t role

  이 변수는 GZLL 네트워크에서의 role을 설정하는것으로 , HOST 또는 DEVICE0 – DEVICE7로 선택할 수 있습니다.

  예:

  ```
  device_t role = DEVICE0;  // 디바이스로 설정되며 device 0으로 세팅됨
  ```

- RFduinoGZLL.begin(role)

  GZLL stack을 시작하고 device_t role 세팅에 의해 설정된 호스트 또는 디바이스로 사용됩니다.

  예:

  ```
  RFduinoGZLL.begin(role); //GZLL stack을 시작.
  ```

- RFduinoGZLL.end()

  GZLL stack을 종료합니다.

  예:

  ```
  RFduinoGZLL.end(); //GZLL stack을 종료.
  ```

- RFduinoGZLL.sendToHost()

  디바이스로 세팅된 블루이노가 호스트로 데이터를 전송합니다.

  예:

  ```
  RFduinoGZLL.sendToHost("Hi Host"); // 호스트에게 hi host라는 문자를 전송함
  ```

- RFduinoGZLL.sendToDevice()

  호스트로 정의된 블루이노가 디바이스에게 데이터를 전송합니다.

  예:

  ```
  RFduinoGZLL.sendToDevice(device, "Hey"); //  디바이스에게 Hey라는 문자를 전송함
  ```

- RFduinoGZLL.onReceive()

RFduinoBLE_onReceive()함수와 비슷한 동작을 하는것으로 수신된 데이터를 처리하는 함수입니다.

예:

```
void RFduinoGZLL_onReceive(device_t device, int rssi, char *data, int len)
{
  char state = data[0];

//호스트가 디바이스에게서 데이터를 수신 받았을때.
  if (device == DEVICE0) // DEVICE0로부터 데이터를 받았는지 확인.
    digitalWrite(green_led, state); //green led에 state 값에 따라 On또는 Off
}
```

- RFduinoGZLL.txPowerLevel

이 변수는 dBm(무선송신단위) 단위로 출력 파워를 조절하는것으로 4dBm 격차로 -20 ~ +4 dBm 사이의 값을 선택할 수 있습니다. (예. -20, -16, -12, -8, -4, 0, +4) - -20이 가장 약한 출력이며 +4가 가장 강한 출력입니다.

예:

```
RFduinoGZLL.txPowerLevel = +4; //전송 전력을 최대 +4dBm으로 설정.
```

[Sleep, Wake]

본 정의는 블루이노의 전력을 최대한 유용하게 이용하기 위한 함수로 sleep과 wake에 대한 정의를 설명합니다.

- RFduino_ULPDelay()

지정된 시간만큼 초저전력으로 유지하게 하여, 전력을 최소화하게 도와줍니다.

```
RFduino_ULPDelay(uint64_t ms);
```

예:

```
RFduino_ULPDelay(350); // 350 밀리시간
RFduino_ULPDelay(SECONDS(350)); //350 초
RFduino_ULPDelay(MINUTES(350)); //350 분
RFduino_ULPDelay(HOURS(10)); // 10 시간
RFduino_ULPDelay(DAY(3)); // 3 일
RFduino_ULPDelay(INFINITE); // BLE 또는 pinWake()에 의해 인터럽트가 걸릴 때까지 초저전력 모드로 유지.
```

- RFduino_pinWake()

  이 함수는 device를 sleep상태에서 깨우기 위하여 특정 핀에 대해서 할당하는 것을 말합니다.

  예:

  pinMode(5, INPUT); // 5번 pin을 input으로 설정.

  RFduino_pinWake(5, HIGH); // device를 wake up하기 위해 5번 pin을 high 신호로 설정.

- RFduino_pinWoke() & RFduino_resetPinWake()

  pin이 wakeup 발생 여부를 확인하는 함수입니다.

  예:

  RFduino_ULPDelay(INFINITE); // ULP를 계속 유지.

  if (RFduino_pinWoke(5)) { //5번 pin이 wake up을 했다면 여기서 어떤 일을 할 수 있음.

  digitalWrite(led, HIGH);

  RFduino_resetPinWake(5); // wakeup을 야기한 pin state를 reset.

  }

  주의: resetPinWake를 하지 않으면, pinWoke loop에 빠지게 됩니다.

- RFduino_pinWakeCallback()

  이 함수는 깨어났을 때 불리어지는 함수를 정의할수 있는 함수입니다. 즉, sleep상태에서 깨어나면 어떤 특별한 함수를 동작하게 만들수 있습니다.

  RFduino_pinWakeCallback(uint32_t ulPin, uint32_t dwWake, pin_callback_t callback) ;

  예:

  pinMode(6, INPUT); // 6번 pin을 INPUT으로 설정.

  RFduino_pinWakeCallback(6, HIGH, myPinCallback); // 6번 pin으로 device를 wakeup하고 "myPinCallback" 함수를 실행하도록 구성.

- RFduino_systemReset()

  시스템을 리셋시킵니다. 재부팅이 됩니다.

  예:

  RFduino_systemReset();

- RFduino_systemOff()

  이 함수는 system off 상태에서 pinWake를 통해 wake할 수 있는 초저전력 상태로 전환시킵니다.

  예:

  RFduino_systemOff();

- RFduino_temperature()

  블루이노의 칩내부 온도 센서로부터 온도를 확인하는 함수입니다.

  RFduino_temperature(int scale)

  예:

  float temp = RFduino_temperature(CELSIUS); //섭씨 온도로 return하고 float형 온도로 저장. or

  float temp = RFduino_temperature(FAHRENHEIT); //화씨 온도로 return하고 float형 온도로 저장

[기타]

- Serial.begin (baud, RX pin, TX pin)

  이 함수는 표준 아두이노 함수이지만, Blueinno2는 UART를 사용 가능한 GPIO중 어느 것에도 매핑할 수 있습니다.

  예:

  1) Serial.begin(9600) ; // UART 사용을 USB Port로 사용하는 함수
     ; PC 모니터에 정보 출력용
  2) Serial.begin(9600, 2, 3); // 사용자가 임의로 GPIO에 할당하는 경우
     ; baud rate을 9600, RX를 GPIO 2, TX를 GPIO 3로 하여 UART를 시작.

- Wire.beginOnPins (SCL pin, SDA pin)

  기본적으로 I2C통신을 SCL => GPIO 6와 SDA => GPIO 5로 설정되어져 있습니다.

  이 함수를 사용하여 통신 포트를 다른것으로 재지정할수 있습니다.

  예:

  Wire.beginOnPins(2, 3); //SCL을 GPIO 2, SDA를 GPIO 3로 하여 I2C interface를 시작.

- Remapping the SPI pins

  SPI 통신을 위하여 기본적으로 MISO(GPIO 3), SCK (GPIO 4), MOSI (GPIO 5), SS/CS (GPIO 6)로 설정되어져 있습니다.

  이 포트가 아닌 다른것으로 설정하고자 한다면 ₩variants₩ Blueinno 폴더에 있는 variant.h 파일을 열고 다음 정의들을 수정하면 됩니다..

  예:

  #define PIN_SPI_SS (6u)

  #define PIN_SPI_MOSI (5u)

  #define PIN_SPI_MISO (3u)

  #define PIN_SPI_SCK (4u)

UART 통신	Serial.begin(baud) ; USB 포트 출력 , PC 와 통신시 사용 Serial.begin(baud, RX pin, TX pin) ; GPIO 핀에 사용자 할당
I2C 통신	Wire.begin() ; 기본설정 SCL = 6 번, SDA = 5 번 Wire.beginOnPins(SCL pin, SDA pin) ; GPIO 핀에 사용자 할당
SPI 통신	기본설정 MISO = 3 번, SCK =4 번 , MOSI =5 번 , SS/CS = 6 번 사용자 할당시는 variant.h 파일을 수정해야 함

[analogReference 함수 (2015년 6월 15일)]

1. 모듈형의 전원(VDD)와 센서의 전원을 같이 사용할 때

- analogReference(VDD_1_3_PS)
- 함수 미 사용시, default

만약) GPIO의 analogRead(3); 3번핀에 물리적 인가전압은 3.6V까지 허용합니다.
analog ADC값을 읽으면, VDD의 1/3전압과 analogRead값의 1/3값을 서로 비교하여 ADC값으로 표시합니다.

(주의사항)

1) VDD전압이 3.6V보다 낮은 3.3V로 구동시, 1.1V를 기준으로 하므로, GPIO인가 되는 신호의 레벨을 더 높게 측정하는 오류가 발생할 수 있습니다.

2) VDD 전압이 2.2V이하에서 GPIO인가 신호의 레벨이 2.4V를 넘으면, 포화되어 정확한 값을 읽을 수가 없습니다.
보통은 GPIO인가 전압을 저항으로 분배하여, 1.5V이하로 인가하는 것을 권장합니다.

2. 모듈형의 전원(VDD)와 센서의 전원을 분리 사용할 때

; 모듈형의 전원(VDD)은 저전력을 위해서, 배터리를 직접 연결해서 사용합니다. 동작 전압은 2.1 ~ 3.6V 입니다.
시간에 따라서, 배터리가 소모되어 입력 전압이 변합니다.

IO 입력단에 연결된 센서의 공급 전원은 일정한 전원 입력 +
전원 공급을 On/Off 스위칭 제어을 위해서
별도의 LDO나 DC/DC 를 이용합니다.

이런 경우에는 아래의 함수를 Setup에 추가해 주어야만
IO 입력단의 센서값을 정확하게 읽을 수 있습니다.

analogReference(VBG); // 칩 내부 LDO를 통해 기준전압(1.2V)을 설정함

// 모듈형의 최대 내압을 3.6V로 공급한다면,
GPIO에 인가되는 전압에 의해서 0~ 3.6V = ADC (0 ~ 1023)로 읽습니다.

(주의사항)

1) VDD전압 보다 GPIO 인가 되는 신호의 레벨을 더 높을 때,
포화되는 증상으로 오류가 발생할 수 있습니다.

- **배터리 전원을 측정하고자 하는 경우**

```
void setup() {
analogReference(VBG); // Sets the Reference to 1.2V band gap
analogSelection(VDD_1_3_PS);   //Selects VDD with 1/3 prescaling as the analog source
Serial.begin(9600);
}

void loop() {
int sensorValue = analogRead(1); // the pin has no meaning, it uses VDD pin
float batteryVoltage = sensorValue * (3.6 / 1023.0); // convert value to voltage
Serial.println(batteryVoltage);
delay(100);
}
```

// Reference 전압은 칩내부의 1.2V를 기준으로 하며, ADC MUX가 GPIO에서 VDD로 전환하여, VDD를 1/3 Prescale하여 전압을 표시함
analogRead(1); // GPIO를 analogRead를 하여도 VDD값을 읽음

(오류사례)
analogSelection(VDD_1_3_PS); // 이 함수만 선언시

⇨ analogReference(VDD_1_3_PS) // 동일한 코딩
analogSelection(VDD_1_3_PS);

// 기준전압 = 인가전압이 모두 VDD의 1/3전압이므로,
항상 3.6V, ADC=1023으로 표시됨

[Interrupt 함수 (2016년 04월 04일)]

attachInterrupt(pin,Callback,state); // int.0 = pin1
int.1 = pin2
int.2 = pin3
int.3 = pin4
(주의) pin0 ~ 7 외의 핀들은 사용 불가

// Interrupt 발생시 Callback() { 함수내에 코드 실행 }

// state : 아직 검토중! 160404
LOW : 실제 동작은 FALLING으로 동작
해당 핀이 HIGH -> LOW 로 될 때
HIGH : 실제 동작은 RISING으로 동작
해당 핀이 LOW -> HIGH로 될 때

RISING / FALLING / CHANGE : 동작하지 않음

detachInterrupt(pin); // Interrupt Disable

[SoftwareSerial 함수 (2015년 10월 26일)]

#include <SoftwareSerial.h> // 헤더 파일 추가

SoftwareSerial (rx_pin, tx_pin) // rx_pin 0 ~ 7 까지, Interrupt pin(0~7) 사용

begin(9600) // 최대 96,00bps까지

[GPIO 드라이브 전류 및 함수 (2015년 11월 28일)]

pinMode(x, OUTPUT) ; function will configure them for High drive
; default, up to 3 pins for 15mA max

pinMode(x, OUTPUT_SOS1) ; function will configure them for Standard drive

[Serial 함수와 BLE 함수 동시 사용 주의 사항 (2015년 11월 28일)]

```
override_uart_limit = true; // 시리얼과 BLE 를 동시 사용을 비추하며,
                // 꼭! 사용시 9600 bps 로 낮추시고, 그래도 응답이 늦으시면
                // 신뢰성 보장은 어려우나 기본 성능이 가능한 이 코드를 삽입!
```