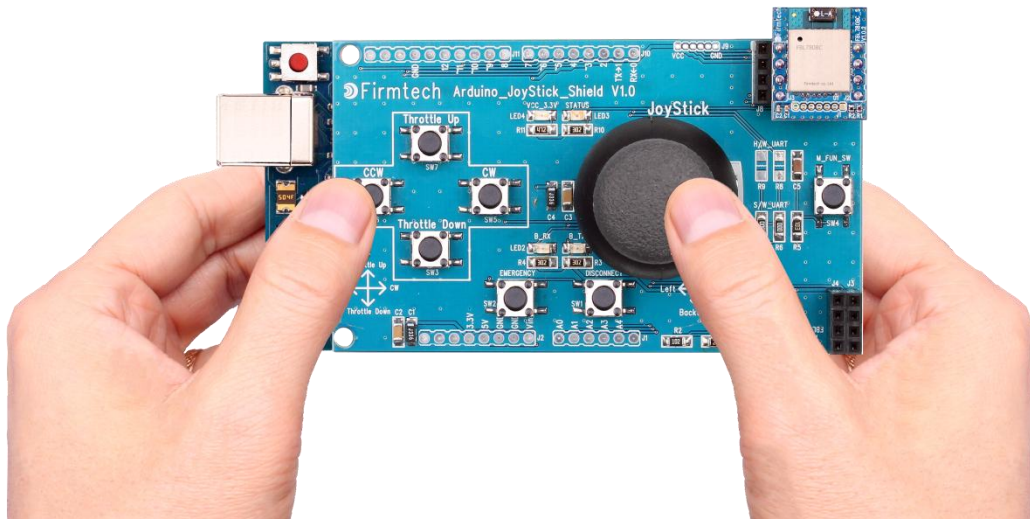


Arduino+Drone Kit

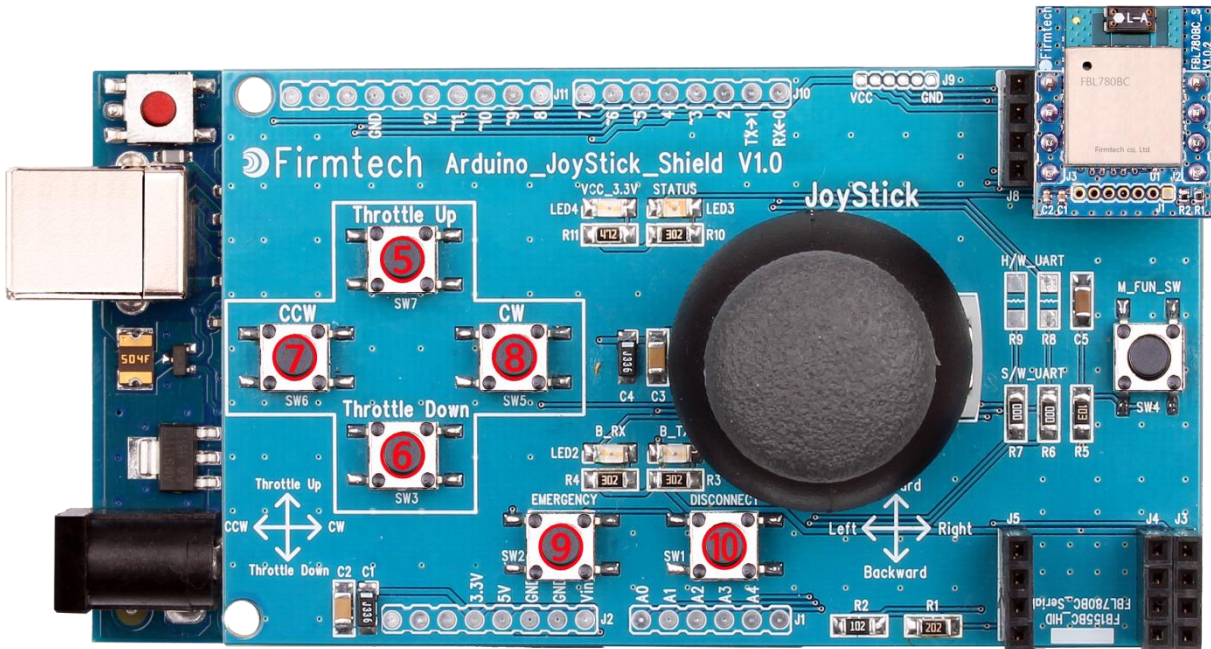
기능별 소스 분류에 대한 상세 설명



기능별 소스 분류에 대한 상세 설명(목차)

No	소스 화일명 (확장자 ino)	소스 세부 설명
01	Joystick_01_Debug_01	시리얼 포트를 통한 메시지 출력 방법 익히기-(1)
02	Joystick_01_Debug_02	시리얼 포트를 통한 메시지 출력 방법 익히기-(2)
03	Joystick_02_Define	Define문 사용법 익히기
04	Joystick_03_Variable	변수 사용법 익히기
05	Joystick_04_String_01	스트링(문자열) 사용법 익히기-(1)
06	Joystick_04_String_02	스트링(문자열) 사용법 익히기-(2)
07	Joystick_05_Pio_01	디지털 입,출력핀 사용법 익히기-(1)
08	Joystick_05_Pio_02	디지털 입,출력핀 사용법 익히기-(2)
09	Joystick_06_Adc_01	아날로그 입력 단자 사용법 익히기-(1)
10	Joystick_06_Adc_02	아날로그 입력 단자 사용법 익히기-(2)
11	Joystick_07_Function_01	함수 사용법 익히기-(1)
12	Joystick_07_Function_02	함수 사용법 익히기-(2)
13	Joystick_08_Checksum_01	데이터 통신 오류 검출에 활용되는 Checksum 사용법 익히기-(1)
14	Joystick_08_Checksum_02	데이터 통신 오류 검출에 활용되는 Checksum 사용법 익히기-(2)
15	Joystick_09_Switch-Case	Switch-case문 사용법 익히기
16	Joystick_10_startsWith	아두이노 내부 함수 사용법 익히기 - 특정 문자열 검색
17	Joystick_11_Serial_01	아두이노 내부 함수 사용법 익히기 - 하드웨어 UART (TX,RX,GND) 설정
18	Joystick_11_Serial_02	아두이노 내부 함수 사용법 익히기 - 소프트웨어 UART (TX,RX,GND) 설정
19	Joystick_12_AT-Command	아두이노로 블루투스 모듈(BLE 모듈) 사용법 익히기-(1)
20	Joystick_13_BT-Connect	아두이노로 블루투스 모듈(BLE 모듈) 사용법 익히기-(2) 드론과의 연결 진행
21	Joystick_14_Check-Message_01	아두이노로 블루투스 모듈(BLE 모듈) 사용법 익히기-(3)
22	Joystick_14_Check-Message_02	아두이노로 블루투스 모듈(BLE 모듈) 사용법 익히기-(4)
23	Joystick_15_DataPacket_01	드론과의 통신에 사용되는 데이터 송,수신 패킷 구조 익히기-(1)
24	Joystick_15_DataPacket_02	드론과의 통신에 사용되는 데이터 송,수신 패킷 구조 익히기-(2)
25	Joystick_15_DataPacket_02-1	드론과의 통신에 사용되는 데이터 송,수신 패킷 구조 익히기-(2)
26	Joystick_15_DataPacket_02-2	드론과의 통신에 사용되는 데이터 송,수신 패킷 구조 익히기-(2)
27	Joystick_15_DataPacket_03	드론과의 통신에 사용되는 데이터 송,수신 패킷 구조 익히기-(3)
28	Joystick_15_DataPacket_04	드론을 제어해 보자 - 드론 전진, 후진
29	Joystick_15_DataPacket_05	드론을 제어해 보자 - 드론 좌측 이동, 우측 이동
30	Joystick_15_DataPacket_06	드론을 제어해 보자 - 드론 좌회전, 우회전
31	Joystick_15_DataPacket_07	드론을 제어해 보자 - 드론 비행고도(높이) 조정

Joystick Shield Description



번호	포트 번호	기능 구분	기 능
⑤	PIO5	Throttle UP	드론을 상승 시킵니다.
⑥	PIO6	Throttle Down	드론을 하강 시킵니다.
⑦	PIO7	CCW	드론을 좌회전 (제자리에서) 시킵니다.
⑧	PIO8	CW	드론을 우회전 (제자리에서) 시킵니다.
⑨	PIO9	Emergency	드론과의 연결 요청시 사용됩니다. 드론과 연결 후에는 응급 버튼으로 사용
⑩	PIO10	Disconnect	드론과 연결 후 연결 해지 버튼으로 사용

1. Jostick_01_Debug_01

▶ 아두이노는 시리얼 포트를 이용한 디버깅 메시지 확인이 쉽고, 용이합니다.

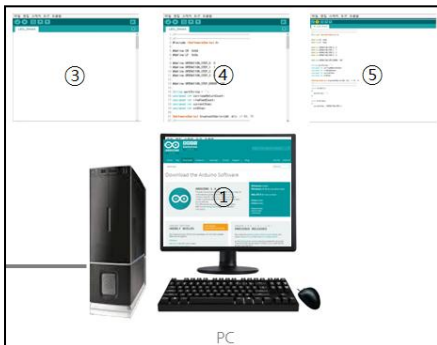
```
void setup()
{
  Serial.begin(9600);
  Serial.println("Debug Test 01");
}

void loop()
{
}
```

"Serial.begin(9600);"만 사용하면 시리얼 포트 사용준비가 완료됩니다.

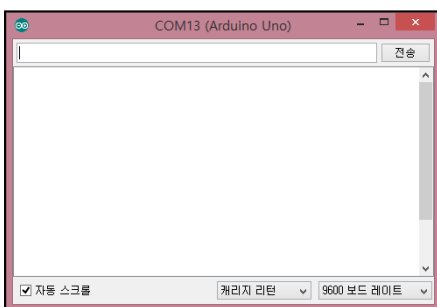
출력하고자 하는 시리얼 메시지를 "Serial.println()"에 입력합니다.

여기서는 "Debug Test 01"을 입력했습니다.



① 아두이노 프로그램으로 Joystick_01_Debug_01.ino를 엽니다.

② 아두이노에 다운로드를 진행합니다.



③ 아두이노 시리얼 창을 실행합니다.

④ 출력되는 시리얼 데이터를 확인합니다.

2. Jostick_01_Debug_02

▶ 디버깅 메시지는 시리얼 메시지 뿐만 아니라 숫자도 확인이 가능합니다.

```
unsigned int i;
//-----
void setup()
{
  Serial.begin(9600);
  Serial.println("Debug Test 02");
}

void loop()
{
  Serial.print(i);
  i++;
  if(i > 10)
  {
    i = 0;
    Serial.println();
    Serial.println("Serial Test");
  }
  delay(300);
}
```

숫자를 저장할 공간으로 'i'를 사용합니다.

"Serial.begin(9600);"만 사용하면 시리얼 포트 사용준비가 완료됩니다.

출력하고자 하는 시리얼 메시지를 "Serial.println()"에 입력합니다.

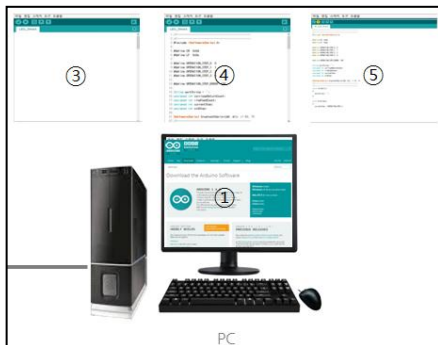
여기서는 "Debug Test 02"를 입력했습니다.

저장된 숫자를 출력하기 위해서 "Serial.print(i)"를 사용합니다.

i는 1씩 증가하도록 합니다.

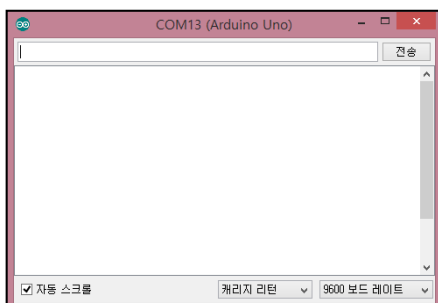
i가 10보다 커지면 i를 0으로 저장하고 "Serial Test"라는 메시지를 출력하도록 합니다.

이 동작은 300ms간격으로 반복됩니다.



① 아두이노 프로그램으로 Joystick_01_Debug_02.ino를 엽니다.

② 아두이노에 다운로드를 진행합니다.



③ 아두이노 시리얼 창을 실행합니다.

④ 출력되는 시리얼 데이터를 확인합니다

3. Joystick_02_Define

- ▶ Define문은 프로그램 작성의 효율을 높이고 소스 분석의 이해도를 높이기 위해 사용합니다.
단순히 숫자 '3'을 사용하는 경우, 이 숫자의 의미 파악을 바로 하는 것은 어렵습니다.
그러나, "현재 진행 중인 3번째 스텝"이라는 의미를 부여한다면 이해도가 높아집니다.

```
#define OPERATION_STEP_0 0
#define OPERATION_STEP_1 1
#define OPERATION_STEP_2 2
#define OPERATION_STEP_3 3
#define OPERATION_STEP_4 4
#define OPERATION_STEP_5 5
#define OPERATION_STEP_6 6
#define OPERATION_STEP_7 7
#define OPERATION_STEP_8 8
#define OPERATION_STEP_9 9
#define OPERATION_STEP_10 10
```

Define문을 사용하는 방법은 왼쪽과 같습니다.

숫자 0을 "OPERATION_STEP_0"으로 정의합니다.
이것은 "진행중인 0번째 스텝"이라는 의미로 생각하시면 됩니다.

숫자 5를 "OPERATION_STEP_5"로 정의합니다.
이것은 "진행중인 5번째 스텝"이라는 의미로 생각하시면 됩니다.

```
void setup()
{
    Serial.begin(9600);
    Serial.println("Define Test");

    Serial.println(OPERATION_STEP_0);
    Serial.println(OPERATION_STEP_1);
    Serial.println(OPERATION_STEP_2);
    Serial.println(OPERATION_STEP_3);
    Serial.println(OPERATION_STEP_4);
    Serial.println(OPERATION_STEP_5);
    Serial.println(OPERATION_STEP_6);
    Serial.println(OPERATION_STEP_7);
    Serial.println(OPERATION_STEP_8);
    Serial.println(OPERATION_STEP_9);
}

void loop()
{
}
```

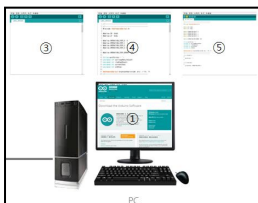
"Serial.begin(9600);"만 사용하면 시리얼 포트 사용준비가 완료됩니다.

출력하고자 하는 시리얼 메시지를 "Serial.println()"에 입력합니다.

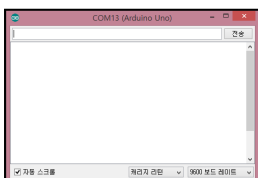
여기서는 "Define Test"를 입력했습니다.

Define문으로 정의된 것을 "Serial.println()"에 입력합니다.

Define문의로 정의된 숫자가 출력되는 것을 확인할 수 있습니다.



- ① 아두이노 프로그램으로 Joystick_02_Define.ino를 엽니다.
- ② 아두이노에 다운로드를 진행합니다.



- ③ 아두이노 시리얼 창을 실행합니다.
- ④ 출력되는 시리얼 데이터 확인합니다.

4. Joystick_03_Variable

▶ 변수는 변화되는 수를 저장하는 공간이며, 하나의 공간에 다른 값의 수를 저장할 수 있습니다.

예) 변수 i에 0이라는 수를 저장하는 경우 : $i = 0$, 이 순간 변수 i는 숫자 0과 같습니다.

변수 i에 5라는 수를 저장하는 경우 : $i = 5$, 이 순간 변수 i는 숫자 5와 같습니다.

```
void setup()
{
  Serial.begin(9600);
  Serial.println("Variable Test");

  for(int i = 0; i < 100; i++)
  {
    Serial.println(i);
  }
}

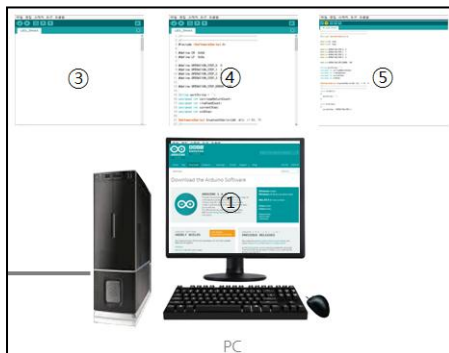
void loop()
{
}
```

"Serial.begin(9600);"만 사용하면 시리얼 포트 사용준비가 완료됩니다.

출력하고자 하는 시리얼 메시지를 "Serial.println()"에 입력합니다.
여기서는 "Variable Test"를 입력했습니다.

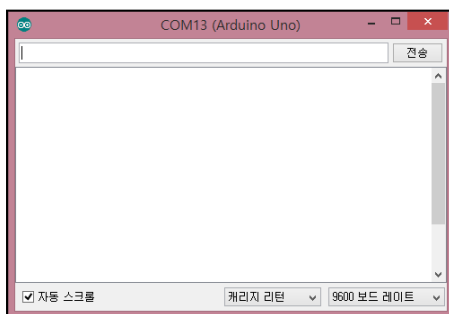
숫자를 저장할 공간으로 'i'를 사용합니다.
숫자는 0에서 99까지 1씩 증가하도록 합니다.
변화된 숫자는 i에 저장됩니다.

저장된 숫자를 출력하기 위해서 "Serial.println(i)"를 사용합니다.



① 아두이노 프로그램으로 Joystick_03_Variable.ino를 엽니다.

② 아두이노에 다운로드를 진행합니다.



③ 아두이노 시리얼 창을 실행합니다.

④ 출력되는 시리얼 데이터를 확인합니다.

5. Joystick_04_String_01

- ▶ 스트링은 문자열을 저장하는 공간입니다. 즉, 문자열을 저장하는 변수이며, 하나의 공간에 다른 문자열을 저장할 수 있습니다.

예) 스트링 변수 st에 "abc"를 저장하는 경우 : st = "abc", 이 순간 스트링 변수 st는 문자열 "abc"와 같습니다.
스트링 변수 st에 "12b"를 저장하는 경우 : st = "12b", 이 순간 스트링 변수 st는 문자열 "12b"와 같습니다.

```
String testString = "";  
//-----  
void setup()  
{  
  Serial.begin(9600);  
  Serial.println("String Test 01");  
  
  testString = "1234567890";  
  Serial.println(testString);  
}  
  
void loop()  
{  
  
}
```

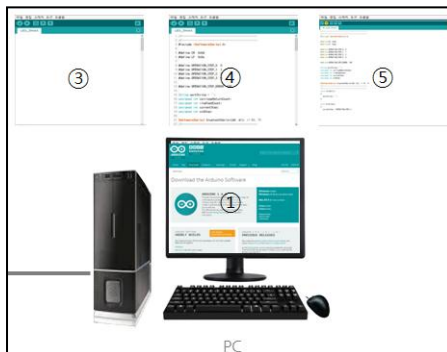
문자 열을 저장할 공간으로 'testString'을 사용합니다.

"Serial.begin(9600);"만 사용하면 시리얼 포트 사용준비가 완료됩니다.

출력하고자 하는 시리얼 메시지를 "Serial.println()"에 입력합니다.
여기서는 "String Test 01"을 입력했습니다.

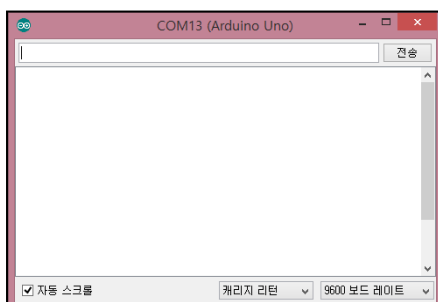
스트링 변수에 "1234567890"을 저장합니다.

스트링 변수를 출력하기 위해서 "Serial.println(testString)"을 사용합니다.



① 아두이노 프로그램으로 Joystick_04_String_01.ino를 엽니다.

② 아두이노에 다운로드를 진행합니다.



③ 아두이노 시리얼 창을 실행합니다.

④ 출력되는 시리얼 데이터 확인합니다.

6. Joystick_04_String_02

```
String testString = "";  
//-----  
void setup()  
{  
  Serial.begin(9600);  
  Serial.println("String Test 02");  
  
  testString = "1234567890";  
  Serial.println(testString);  
  
  testString = "abc";  
  Serial.println(testString);  
}  
  
void loop()  
{  
  
}
```

문자열을 저장할 공간으로 'testString'을 사용합니다.

"Serial.begin(9600);"만 사용하면 시리얼 포트 사용준비가 완료됩니다.

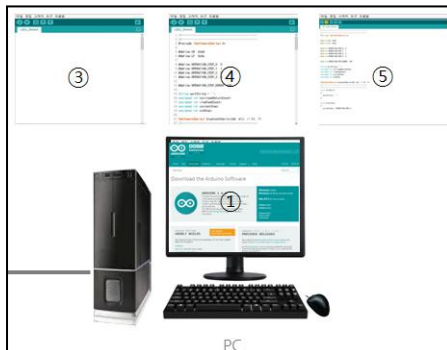
출력하고자 하는 시리얼 메시지를 "Serial.println()"에 입력합니다.
여기서는 "String Test 02"를 입력했습니다.

스트링 변수에 "1234567890"을 저장합니다.

스트링 변수를 출력하기 위해서 "Serial.println(testString)"을 사용합니다.

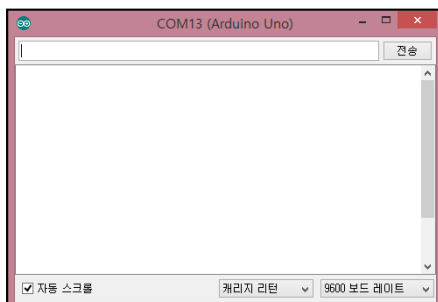
스트링 변수에 "abc"를 저장합니다.

스트링 변수를 출력하기 위해서 "Serial.println(testString)"을 사용합니다.



① 아두이노 프로그램으로 Joystick_04_String_02.ino를 엽니다.

② 아두이노에 다운로드를 진행합니다.



③ 아두이노 시리얼 창을 실행합니다.

④ 출력되는 시리얼 데이터 확인합니다.

7. Joystick_05_Pio_01

- ▶ 디지털 포트는 전기적 신호를 Low/High로 보내거나 Low/High로 입력된 상태를 체크합니다.
 펌테크 "조이스틱 쉴드"는 입력 상태 체크만 가능합니다.
 펌테크 "조이스틱 쉴드"는 High상태에서 Low상태로 변화된 경우를 체크합니다.

```
void setup()
{
  Serial.begin(9600);
  Serial.println("PIO Test 01");

  for(int i = 5; i < 11; i++)
  {
    pinMode(i, INPUT);
    digitalWrite(i, HIGH);
  }
}

void loop()
{
  if(!digitalRead(9))
  {
    Serial.println("Press 9");
  }
}
```

"Serial.begin(9600);"만 사용하면 시리얼 포트 사용준비가 완료됩니다.

출력하고자 하는 시리얼 메시지를 "Serial.println()"에 입력합니다.
여기서는 "PIO Test 01"을 입력했습니다.

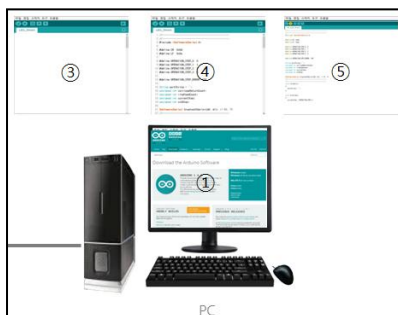
5~10번까지의 6개의 디지털 포트를 사용합니다.

pinMode(i, INPUT)을 사용하여 6개의 디지털 포트를 입력으로 설정합니다.

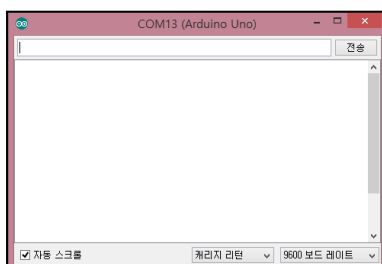
digitalWrite(i, HIGH)를 사용하여 6개의 디지털 포트의 초기값을 High 상태로 만듭니다.

"if(!digitalRead(9))"을 사용하여 디지털 9번 포트가 High 상태에서 Low 상태로 변경되었는지 체크합니다.

9번 포트가 Low 상태로 변경된 경우, "Serial.println("Press 9")"을 이용하여 메시지를 출력합니다.



- ① "조이스틱 쉴드"를 아두이노에 장착합니다.
- ② 아두이노 프로그램으로 Joystick_05_Pio_01.ino를 엽니다.
- ③ 아두이노에 다운로드를 진행합니다.



- ④ 아두이노 시리얼 창을 실행합니다.
- ⑤ PIO 9번 스위치를 누릅니다.
- ⑥ 출력되는 시리얼 데이터 확인합니다.

8. Joystick_05_Pio_02

```
void setup()
{
  Serial.begin(9600);
  Serial.println("PIO Test 02");

  for(int i = 5; i < 11; i++)
  {
    pinMode(i, INPUT);
    digitalWrite(i, HIGH);
  }
}

void loop()
{
  if(!digitalRead(5))
  {
    Serial.println("Press 5");
  }
  if(!digitalRead(6))
  {
    Serial.println("Press 6");
  }
  if(!digitalRead(7))
  {
    Serial.println("Press 7");
  }
  if(!digitalRead(8))
  {
    Serial.println("Press 8");
  }
  if(!digitalRead(9))
  {
    Serial.println("Press 9");
  }
  if(!digitalRead(10))
  {
    Serial.println("Press 10");
  }
}
```

"Serial.begin(9600);"만 사용하면 시리얼 포트 사용준비가 완료됩니다.

출력하고자 하는 시리얼 메시지를 "Serial.println()"에 입력합니다.
여기서는 "PIO Test 02"를 입력했습니다.

5~10번까지의 6개의 디지털 포트를 사용합니다.

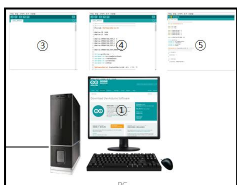
pinMode(i, INPUT)을 사용하여 6개의 디지털 포트를 입력으로 설정합니다.

digitalWrite(i, HIGH)를 사용하여 6개의 디지털 포트의 초기값을 High 상태로 만듭니다.

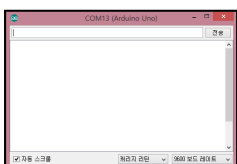
"if(!digitalRead(5))"를 사용하여 디지털 5번 포트가 High 상태에서 Low 상태로 변경되었는지 체크합니다.

5번 포트가 Low 상태로 변경된 경우, "Serial.println("Press 5")"를 이용하여 메시지를 출력합니다.

6개의 디지털 포트를 모두 체크합니다.



- ① "조이스틱 쉴드"를 아두이노에 장착합니다.
- ② 아두이노 프로그램으로 Joystick_05_Pio_02.ino 엽니다.
- ③ 아두이노에 다운로드를 진행합니다.



- ④ 아두이노 시리얼 창을 실행합니다.
- ⑤ PIO 5~10번 스위치를 누릅니다.
- ⑥ 출력되는 시리얼 데이터 확인합니다.

9. Joystick_06_Adc_01

- ▶ 아두이노의 아날로그 포트는 0 ~ 5V 사이의 연속적인 신호의 입력 체크가 가능한 포트입니다.
아두이노의 아날로그 포트는 연속적인 신호의 순간 값을 읽는 포트입니다.
아두이노의 아날로그 포트는 연속적인 신호의 값을 읽어 0~1024의 디지털 값(순간 값)으로 변경합니다.

아두이노의 아날로그 포트에 0V가 입력된 경우, 아두이노는 디지털 값 0을 출력합니다.
아두이노의 아날로그 포트에 2.5V가 입력된 경우, 아두이노는 디지털 값 512를 출력합니다.
아두이노의 아날로그 포트에 5V가 입력된 경우, 아두이노는 디지털 값 1024를 출력합니다.

"조이스틱 쉴드" 조정 장치의 좌우(roll) 이동은 아두이노의 아날로그 4번 포트에 연결되어 있습니다.

"Serial.begin(9600);"만 사용하면 시리얼 포트 사용준비가 완료됩니다.

```
void setup()
{
  Serial.begin(9600);
  Serial.println("Adc Test 01");
}

void loop()
{
  unsigned int adc4 = analogRead(4);

  Serial.println(adc4);
  delay(500);
}
```

출력하고자 하는 시리얼 메시지를 "Serial.println()"에 입력합니다.
여기서는 "Adc Test 01"을 입력했습니다.

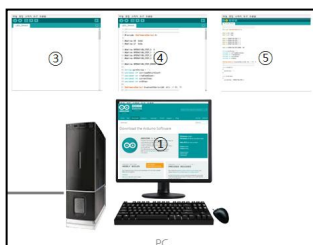
analogRead(4)를 사용하여 아날로그 값을 읽습니다.

'4'는 아날로그 포트 4를 의미합니다.

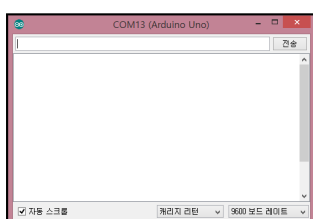
읽어 온 아날로그 값을 변수 adc4에 저장합니다.

저장된 아날로그 값을 "Serial.println(adc4);"를 이용하여
시리얼 데이터로 출력합니다.

이 동작은 500ms간격으로 반복됩니다.



- ① "조이스틱 쉴드"를 아두이노에 장착합니다.
- ② 아두이노 프로그램으로 Joystick_06_Adc_01.ino를 엽니다.
- ③ 아두이노에 다운로드를 진행합니다.



- ④ 아두이노 시리얼 창을 실행합니다.
- ⑤ 조이스틱을 좌/우로 변경합니다.
- ⑥ 출력되는 시리얼 데이터 확인합니다.

10. Joystick_06_Adc_02

- ▶ "조이스틱 쉴드" 조정장치의 좌우(roll) 이동은 아두이노의 아날로그 4번 포트에 연결되어 있습니다.
"조이스틱 쉴드" 조정장치의 전후(pitch) 이동은 아두이노의 아날로그 5번 포트에 연결되어 있습니다.

```
void setup()
{
  Serial.begin(9600);
  Serial.println("Adc Test 02");
}

void loop()
{
  unsigned int adc4 = analogRead(4);
  unsigned int adc5 = analogRead(5);

  Serial.print("adc4 = ");
  Serial.print(adc4);
  Serial.print("   adc5 = ");
  Serial.println(adc5);
  delay(500);
}
```

"Serial.begin(9600);"만 사용하면 시리얼 포트 사용준비가 완료됩니다.

출력하고자 하는 시리얼 메시지를 "Serial.println()"에 입력합니다.
여기서는 "Adc Test 02"를 입력했습니다.

analogRead(4)를 사용하여 아날로그 값을 읽습니다.
'4'는 아날로그 포트 4를 의미합니다.
읽어 온 아날로그 값을 변수 adc4에 저장합니다.

analogRead(5)를 사용하여 아날로그 값을 읽습니다.
'5'는 아날로그 포트 5를 의미합니다.
읽어 온 아날로그 값을 변수 adc5에 저장합니다.

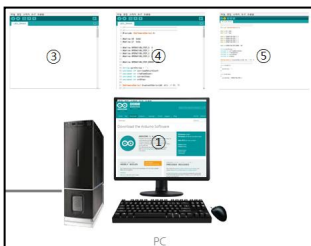
저장된 아날로그 값을 "Serial.println();"을 이용하여 시리얼 데이터로 출력합니다.

2개의 아날로그 포트를 구분하기 위해서 시리얼 포트 "adc4 = "를 출력한 후 adc4의 값을 출력하고, "adc5 = "를 출력한 후 adc5의 값을 출력합니다.

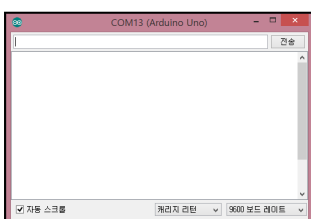
이 동작은 500ms간격으로 반복됩니다.

※ Serial.println();는 줄 바꿈이 발생합니다. 다음 데이터가 아래 줄에 출력됩니다.

※ Serial.print();는 줄 바꿈이 발생되지 않습니다. 다음 데이터가 옆에 출력됩니다.



- ① "조이스틱 쉴드"를 아두이노에 장착합니다.
- ② 아두이노 프로그램으로 Joystick_06_Adc_02.ino를 엽니다.
- ③ 아두이노에 다운로드를 진행합니다.



- ④ 아두이노 시리얼 창을 실행합니다.
- ⑤ 조이스틱을 좌/우 또는 상/하로 변경합니다.
- ⑥ 출력되는 시리얼 데이터 확인합니다.

11. Joystick_07_Function_01

▶ "함수"는 프로그램 작성의 효율을 높이고 소스 분석의 이해도를 높이기 위해 사용합니다.

특별한 기능을 수행하는 부분을 따로 모아 놓으면, 효율과 이해도가 높아집니다.

특별한 기능이 필요한 경우, 기능을 모아 놓은 함수를 호출한다면 프로그램 작성의 효율이 높아집니다.

특별한 기능에 알맞은 이름으로 함수를 정의한 경우, 함수의 이름 만으로도 그 특별한 기능이 어떤 기능을 수행하는지 파악이 가능하여 소스 분석의 이해도가 높아집니다.

```
void disp11111()
{
    Serial.println("11111");
}
//-----
void setup()
{
    Serial.begin(9600);
    Serial.println("Function Test 01");

    disp11111();
}

void loop()
{
}
```

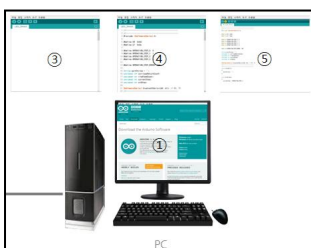
"disp11111()"이란 이름으로 함수를 생성합니다.

이 함수는 시리얼 포트로 "11111"을 출력하는 기능을 합니다.

"Serial.begin(9600);"만 사용하면 시리얼 포트 사용준비가 완료됩니다.

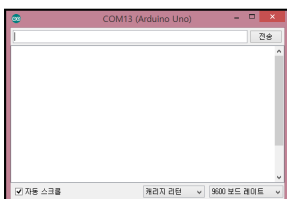
출력하고자 하는 시리얼 메시지를 "Serial.println()"에 입력합니다.
여기서는 "Function Test 01"을 입력했습니다.

작성한 함수를 실행하기 위해 함수의 이름 "disp11111()"을 이용하여
함수를 호출합니다.



① 아두이노 프로그램으로 Joystick_07_Function_01.ino를 엽니다.

② 아두이노에 다운로드를 진행합니다.



③ 아두이노 시리얼 창을 실행합니다.

④ 출력되는 시리얼 데이터 확인합니다.

12. Joystick_07_Function_02

```
void disp11111()
{
    Serial.println("11111");
}

void calculation()
{
    int a = 1;
    int b = 2;
    int c = 0;

    c = a+b;
    Serial.print("a+b=");
    Serial.println(c);
}
//-----
void setup()
{
    Serial.begin(9600);
    Serial.println("Function Test 02");

    disp11111();
    calculation();
}

void loop()
{
}
```

"disp11111()"이란 이름으로 함수를 생성합니다.

이 함수는 시리얼 포트에 "11111"을 출력하는 기능을 합니다.

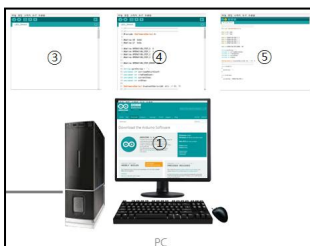
"calculation()"이란 이름으로 함수를 생성합니다.

이 함수는 변수 a에 1을 저장하고, 변수 b에 2를 저장, a와 b를 더해서 변수 c에 저장하고 시리얼로 결과값을 출력하는 기능을 합니다.

"Serial.begin(9600);"만 사용하면 시리얼 포트 사용준비가 완료됩니다.

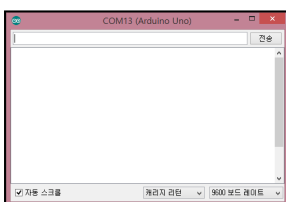
출력하고자 하는 시리얼 메시지를 "Serial.println()"에 입력합니다.
여기서는 "Function Test 02"를 입력했습니다.

작성한 함수를 실행하기 위해 함수의 이름 "disp11111()"와 "calculation()"을 이용하여 함수를 호출합니다.



① 아두이노 프로그램으로 Joystick_07_Function_02.ino를 엽니다.

② 아두이노에 다운로드를 진행합니다.



③ 아두이노 시리얼 창을 실행합니다.

④ 출력되는 시리얼 데이터 확인합니다.

13. Joystick_08_Checksum_01

▶ 데이터를 전송할 때 오류를 검출하기 위해 checksum을 이용합니다.

checksum은 데이터의 정확성을 검사하기 위한 용도로 입력 데이터나 전송 데이터의 맨 마지막에 앞서 보낸 모든 데이터를 다 합한 합계를 따로 보내는 것입니다.

데이터를 받아들이는 측에서는 하나씩 받아들여 합산한 다음 이를 최종적으로 들어온 검사 합계와 비교하여 착오가 있는지를 점검합니다.

```
unsigned char data1 = 0x00;
unsigned char data2 = 0x00;
unsigned char data3 = 0x00;
unsigned char data4 = 0x00;
unsigned char data5 = 0x00;
unsigned char data6 = 0x00;
unsigned char checkSum = 0;
//-----
void setup()
{
  Serial.begin(9600);
  Serial.println("Checksum Test 01");

  for(data1 = 0; data1 < 20; data1++)
  {
    checkSum = data1 + data2 + data3 + data4 + data5 + data6;
    checkSum = checkSum & 0x00ff;
    Serial.println(String(checkSum, HEX));
    delay(100);
  }
}

void loop()
{
}
```

송신 데이터로 data1 ~ data6까지 변수를 지정합니다.

결과 저장 변수로 checkSum을 지정합니다.

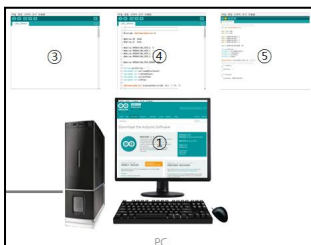
"Serial.begin(9600);"만 사용하면 시리얼 포트 사용준비가 완료됩니다.

출력하고자 하는 시리얼 메시지를 "Serial.println()"에 입력합니다.

여기서는 "Checksum Test 01"을 입력했습니다.

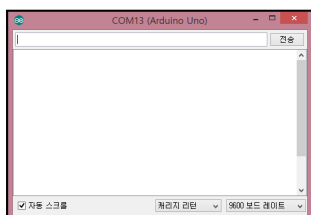
체크섬 값의 변화를 알아보기 위해 모든 데이터를 0으로 하고 data1만 0부터 19까지 변화시킵니다.

계산된 체크섬 값을 시리얼 포트에 출력시킵니다.



① 아두이노 프로그램으로 Joystick_08_Checksum_01.ino를 엽니다.

② 아두이노에 다운로드를 진행합니다.



③ 아두이노 시리얼 창을 실행합니다.

④ 출력되는 시리얼 데이터 확인합니다.

14. Joystick_08_Checksum_02

```
unsigned char data1 = 0xa1;
unsigned char data2 = 0x64;
unsigned char data3 = 0x64;
unsigned char data4 = 0x64;
unsigned char data5 = 0x78;
unsigned char data6 = 0x01;
unsigned char checkSum = 0;
//-----
void setup()
{
  Serial.begin(9600);
  Serial.println("Checksum Test 02");
}

void loop()
{
  checkSum = data1 + data2 + data3 + data4 + data5 + data6;
  checkSum = checkSum & 0x00ff;
  Serial.println(String(checkSum, HEX));
  delay(500);
}
```

각각의 data에 값을 넣고 Checksum을 확인합니다..

"Serial.begin(9600);"만 사용하면 시리얼 포트 사용준비가 완료됩니다.

출력하고자 하는 시리얼 메시지를 "Serial.println()"에 입력합니다.

여기서는 "Checksum Test 02"를 입력했습니다.

정해진 data를 이용하여 계산된 체크섬 값을 시리얼 포트로 출력시킵니다.

이 동작은 500ms간격으로 반복됩니다.

Checksum은 data1, data2, data3, data4, data5, data6의 Hex 값을 더하고, 더한 값과 0x00ff Hex 값을 비트 논리곱 하는 방식으로 구성이 되어있습니다.

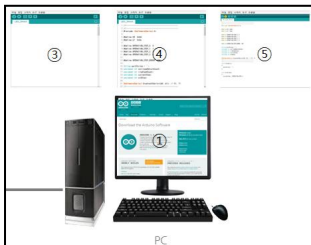
예) data1 = 0xa1, data2 = 0x64, data3 = 0x64, data4 = 0x64, data5 = 0x50, data6 = 0x05 일 경우

$$\text{checkSum} = 0xa1 + 0x64 + 0x64 + 0x64 + 0x50 + 0x05$$

$$= 0x0222$$

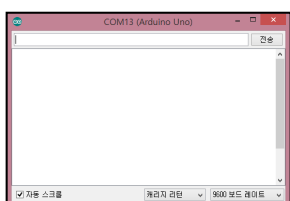
$$\text{checkSum} = 0x0222 \& 0x00ff$$

$$= 0x0022 \text{의 값을 가지게 됩니다.}$$



① 아두이노 프로그램으로 Joystick_08_Checksum_02.ino를 엽니다.

② 아두이노에 다운로드를 진행합니다.



③ 아두이노 시리얼 창을 실행합니다.

④ 출력되는 시리얼 데이터 확인합니다.


15. Joystick_09_Switch-Case

- ▶ 주어진 조건에 따라 실행되는 부분을 다르게 하고자 하는 경우 **"Switch ~ Case 문"**을 사용합니다.
조건은 "switch(xx)"속에 넣고, 실행되는 사항은 "case xx: ~ break;"에 구성합니다.

```
switch(currentStep)
{
    case 1:
        break;
    case 5:

        break;
    default:

        break;
}
```




조건인 "currentStep"의 값에 따라 "case 1: ~ break"가 실행되거나,
"case 5: ~ break"가 실행됩니다.

"Switch ~ Case문" 실행되기 전에 "currentStep = 1"이 되면,
"case 1: ~ break;"가 실행됩니다.

```
switch(currentStep)
{
    case 1:

        break;
    case 5:
        break;
    default:

        break;
}
```



"Switch ~ Case문" 실행되기 전에 "currentStep = 5"가 되면,
"case 5: ~ break;"가 실행됩니다.

```
switch(currentStep)
{
    case 1:

        break;
    case 5:

        break;
    default:
        break;
}
```



"Switch ~ Case문" 실행되기 전에 "currentStep = 3"이 되면,
"default: ~ break;"가 실행됩니다.

```

unsigned int currentStep;
//-----
void setup()
{
  Serial.begin(9600);
  Serial.println("Switch-Case Test");

  currentStep = 1;
}

void loop()
{
  switch(currentStep)
  {
    case 1:
      Serial.println("case 1 Operation");
      currentStep = 5;
      delay(500);
      break;
    case 5:
      Serial.println("case 5 Operation");
      currentStep++;
      delay(500);
      break;
    default:
      Serial.println("Default Operation");
      delay(500);
      break;
  }
}

```

조건 저장 변수로 "currentStep"을 사용합니다.

"Serial.begin(9600);"만 사용하면 시리얼 포트 사용준비가 완료됩니다.

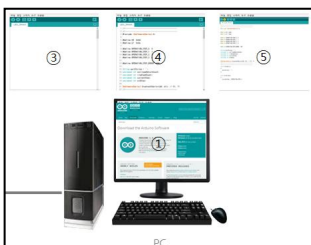
출력하고자 하는 시리얼 메시지를 "Serial.println()"에 입력합니다. 여기서는 "Switch-Case Test"를 입력했습니다.

시작하기 전에 "currentStep = 1"을 진행합니다.

"currentStep = 1"에 의해 "case 1:"부분이 실행됩니다. 시리얼로 "case 1 Operation" 출력이 진행됩니다. "currentStep = 5" 설정이 진행됩니다. 약 500ms 대기합니다. "break;"에 의해 "Switch ~ Case"문이 종료됩니다.

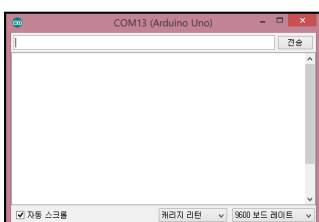
"currentStep = 5"에 의해 "case 5:"부분이 실행됩니다. 시리얼로 "case 5 Operation" 출력이 진행됩니다. "currentStep++" 진행 ("currentStep = 6"이 됩니다.) 약 500ms 대기합니다. "break;"에 의해 "Switch ~ Case"문이 종료됩니다.

"currentStep = 6(currentStep++)"에 의해 "default:"부분이 실행됩니다. 시리얼로 "Default Operation" 출력이 진행됩니다. 약 500ms 대기합니다. "break;"에 의해 "Switch ~ Case"문이 종료됩니다. "currentStep = 6"에 의해 "default:"부분이 계속 실행됩니다.



① 아두이노 프로그램으로 Joystick_09_Switch-Case.ino를 엽니다.

② 아두이노에 다운로드를 진행합니다.



③ 아두이노 시리얼 창을 실행합니다.

④ 출력되는 시리얼 데이터 확인합니다.

16. Joystick_10_startsWith

- ▶ 시리얼 데이터에서 특별한 문자열을 찾고자 하는 경우 아두이노에서 제공하는 내부 함수를 사용하면 쉽게 찾을 수 있습니다.

"startsWith()" → 지정한 문자열로 시작되는 경우, 1(참)을 반환합니다.

:startsWith("123")은 비교하는 시리얼 데이터가 "123"으로 시작하는 경우 1(참)을 반환합니다.

"endsWith()" → 지정한 문자열로 끝나는 경우, 1(참)을 반환합니다.

:endsWith("890")은 비교하는 시리얼 데이터가 "890"으로 끝나는 경우 1(참)을 반환합니다.

"length()" → 문자열의 길이를 반환합니다.

:length()는 비교하는 시리얼 데이터의 길이를 반환합니다.

```
String testString = "1234567890";
//-----
void setup()
{
  Serial.begin(9600);
  Serial.println("startsWith Test");

  Serial.print("Length: ");
  Serial.println(testString.length());

  if(testString.startsWith("123"))
  {
    Serial.println("Start OK");
  }

  if(testString.endsWith("890"))
  {
    Serial.println("End OK");
  }
}

void loop()
{
}
```

문자열 저장 변수로 testString을 사용합니다.

testString에 "1234567890"을 저장합니다.

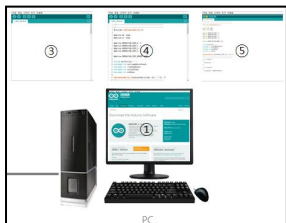
"Serial.begin(9600);"만 사용하면 시리얼 포트 사용준비가 완료됩니다.

출력하고자 하는 시리얼 메시지를 "Serial.println()"에 입력합니다.
여기서는 "startsWith Test"를 입력했습니다.

testString 변수에 저장된 문자열의 길이를 반환하기 위해
"testString.length();"를 사용합니다.
반환된 길이를 "Serial.println()"을 사용하여 출력합니다.

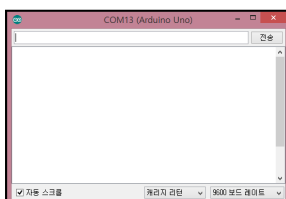
testString 변수가 "123"으로 시작하는지 알아보기 위해
testString.startsWith("123")을 사용합니다.
1(참)이 반환된 경우 시리얼 메시지를 출력합니다.

testString 변수가 "890"으로 끝나는지 알아보기 위해
testString.endsWith("890")을 사용합니다.
1(참)이 반환된 경우 시리얼 메시지를 출력합니다.



① 아두이노 프로그램으로 Joystick_10_startsWith.ino를 엽니다.

② 아두이노에 다운로드를 진행합니다.



③ 아두이노 시리얼 창을 실행합니다.

④ 출력되는 시리얼 데이터 확인합니다.

17. Joystick_11_Serial_01

- ▶ 아두이노는 쉽게 사용할 수 있는 하드웨어 시리얼 포트가 있습니다.

하드웨어 시리얼 포트를 사용하고자 하는 경우 아두이노에서 제공하는 내부 함수를 사용합니다.

"Serial.begin()" → 통신속도를 설정하고, 하드웨어 시리얼 포트를 사용 가능하도록 합니다.

: Serial.begin(9600)은 통신속도 9600으로 하드웨어 시리얼 포트를 사용 가능하도록 합니다.

"Serial.println()" → 출력하고자 하는 메시지를 하드웨어 시리얼 포트에 출력이 가능하도록 합니다.

: Serial.println("Test")는 하드웨어 시리얼 포트에 "Test"라는 메시지를 출력합니다.

"Serial.available()" → 하드웨어 시리얼 포트에 입력된 데이터가 있는 경우 1(참)을 반환합니다.

"Serial.read()" → 하드웨어 시리얼 포트에 입력된 데이터를 읽어옵니다.

"Serial.write()" → 하드웨어 시리얼 포트에 시리얼 데이터를 출력합니다.

```
void setup()
{
  Serial.begin(9600);
  Serial.println("HWSerial Test");
}

void loop()
{
  if(Serial.available())
  {
    Serial.write(Serial.read());
  }
}
```

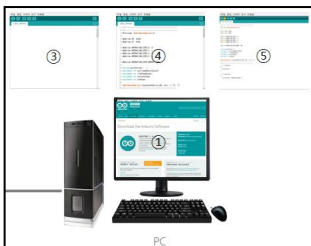
"Serial.begin(9600);"만 사용하면 시리얼 포트 사용준비가 완료됩니다.

출력하고자 하는 시리얼 메시지를 "Serial.println()"에 입력합니다.

여기서는 "HWSerial Test"를 입력했습니다.

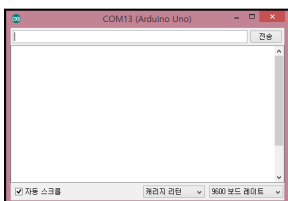
시리얼 창에 데이터를 입력하면 "Serial.available()"이 1(참)을 반환하고, "Serial.read()"에 의해 입력된 데이터를 읽어옵니다. 읽어온 데이터는 "Serial.write()"에 의해 시리얼 창으로 출력됩니다.

"시리얼 창의 입력 방법"은 시리얼 창의 설정에 따라 다르게 동작됩니다.



- ① 아두이노 프로그램으로 Joystick_11_Serial_01.ino를 엽니다.

- ② 아두이노에 다운로드를 진행합니다.



- ③ 아두이노 시리얼 창을 실행합니다.

- ④ 출력되는 시리얼 데이터 확인합니다.

- ⑤ 아두이노 시리얼 창에 시리얼 데이터를 입력하고 출력되는 시리얼 데이터를 확인합니다.

18. Joystick_11_Serial_02

- ▶ "아두이노 우노"는 1개의 하드웨어 시리얼 포트가 있습니다. 필요한 경우 소프트웨어 시리얼 포트를 추가하여 2개의 시리얼 포트를 사용합니다.
- ▶ 소프트웨어 시리얼 포트를 사용하고자 하는 경우 아두이노에서 제공하는 외부 함수를 사용하면 쉽게 소프트웨어 시리얼 포트를 사용할 수 있습니다.
- ▶ 아두이노에서 제공하는 외부 함수를 사용하기 위해서는 소스 상에서 라이브러리를 추가해야 합니다.
추가 방법은 "#include <SoftwareSerial.h>" 입니다.
추가된 라이브러리를 사용하기 위해서는 "인자"를 생성해야 합니다.
인자 생성 방법은 "SoftwareSerial bleSerial(A0,A1);"입니다. 이것은 A0과 A1 포트로 "bleSerial" 이라는 인자를 사용하여 소프트웨어 시리얼을 사용하겠다고 알리는 것입니다.
- ▶ 기본적인 사용 방법은 하드웨어 시리얼 포트와 유사합니다.

```
#include <SoftwareSerial.h>

SoftwareSerial bleSerial(A0, A1); // RX, TX
//-----
void setup()
{
  Serial.begin(9600);
  bleSerial.begin(9600);
  Serial.println("HW-SW Serial Test");
}

void loop()
{
  if(bleSerial.available())
    Serial.write(bleSerial.read());
  if(Serial.available())
    bleSerial.write(Serial.read());
}
```

소프트웨어 시리얼 사용을 위해 "라이브러리"를 추가합니다.

라이브러리를 사용하기 위해 "인자"를 생성합니다.

"Serial.begin(9600);"만 사용하면 하드웨어 시리얼 포트 사용준비가 완료됩니다.

"bleSerial.begin(9600);"만 사용하면 소프트웨어 시리얼 포트 사용준비가 완료됩니다.

하드웨어 시리얼 포트에 출력하고자 하는 시리얼 메시지를 "Serial.println()"에 입력합니다.
여기서는 "HW-SW Serial Test"를 입력했습니다.

소프트웨어 시리얼 포트에 데이터가 입력되면 "bleSerial.available()"이 1(참)을 반환하고, "bleSerial.read()"에

의해 소프트웨어 시리얼 포트에 입력된 데이터를 읽어옵니다. 읽어온 데이터는 "Serial.write()"에 의해 하드웨어 시리얼 포트에 출력됩니다.

하드웨어 시리얼 포트에 데이터가 입력되면 "Serial.available()"이 1(참)을 반환하고, "Serial.read()"에 의해 하드웨어 시리얼 포트에 입력된 데이터를 읽어옵니다. 읽어온 데이터는 "bleSerial.write()"에 의해 소프트웨어 시리얼 포트에 출력됩니다.

이 기능은 아두이노를 가운데 두고 하드웨어 시리얼 포트를 이용하여 명령어를 입력하면 아두이노가 입력된 명령어를 소프트웨어 시리얼 포트에 출력하고, 소프트웨어 시리얼 포트에 입력된 응답 값을 아두이노가 하드웨어 시리얼 포트에 출력하는 것입니다.

아두이노 프로그램 시리얼 창에 AT Command를 입력하면, 아두이노는 입력 된 AT Command를 소프트웨어 시리얼 포트를 통해 BLE 장치(FBL780_Serial)에 전달합니다.

소프트웨어 시리얼 포트를 통해 BLE 장치(FBL780_Serial)가 응답 값을 전달하면, 아두이노는 입력된 응답 값을 하드웨어 시리얼 포트를 통해 아두이노 IDE 시리얼 창에 전달합니다.

아두이노 "조이스틱 쉴드"에 사용하는 BLE 장치(FBL780_Serial)는 Central Role으로 동작해야 합니다.

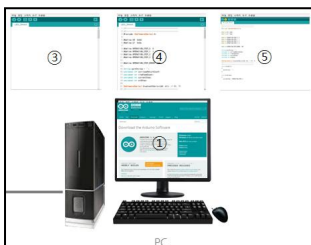
- ▶ BLE 장치가 Central Role으로 동작하는지 확인하는 방법은 아래와 같습니다.
 - (1) 아두이노 시리얼 창에 "AT+GETROLE" 명령어를 입력하시면 됩니다.
 - (2) 응답값으로 "C" 가 송신되면 BLE 장치가 Central Role으로 동작하는 것입니다.
- ▶ BLE 장치가 Central Role으로 동작하지 않는 경우 설정하는 방법은 아래와 같습니다.
 - (1) 아두이노 시리얼 창에 "AT+SETROLEC" 명령어를 입력하시면 됩니다.
 - (2) 응답값으로 "OK" 가 송신되면 아두이노 시리얼 창에 "ATZ" 명령어를 입력하시면 됩니다.

아두이노 "조이스틱 쉴드"에 사용하는 BLE 장치(FBL780_Serial)는 SMODE가 0으로 동작해야 합니다.

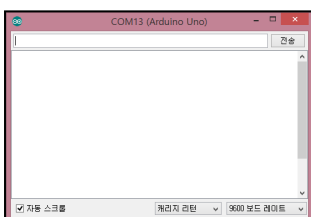
- ▶ BLE 장치가 SMODE 0으로 동작하는지 확인하는 방법은 아래와 같습니다.
 - (1) 아두이노 시리얼 창에 "AT+GETSMODE" 명령어를 입력하시면 됩니다.
 - (2) 응답값으로 "0" 이 송신되면 BLE 장치가 SMODE 0 으로 동작을 하는 것입니다.
- ▶ BLE 장치가 SMODE 0으로 동작하지 않을 경우 설정하는 방법은 아래와 같습니다.
 - (1) 아두이노 시리얼 창에 "AT+SETSMODE0" 명령어를 입력하시면 됩니다.
 - (2) 응답값으로 "OK" 가 송신되면 아두이노 시리얼 창에 "ATZ" 명령어를 입력하시면 됩니다.

※ 소프트웨어 시리얼 포트는 높은 통신속도에서 오동작이 되므로, 가능하면 낮은 통신 속도로 사용해야 합니다.

"아두이노 프로그램 시리얼 창의 입력 방법"은 시리얼 창의 설정에 따라 다르게 동작됩니다.



- ① "조이스틱 쉴드"를 아두이노에 장착합니다.
- ② BLE 장치를 "조이스틱 쉴드"에 장착합니다.
- ③ 아두이노 프로그램으로 Joystick_11_Serial_02.ino를 엽니다.
- ④ 아두이노에 다운로드를 진행합니다.



- ④ 아두이노 시리얼 창을 실행합니다.
- ⑤ 출력되는 시리얼 데이터 확인합니다.
- ⑥ AT Command 입력 & 응답 값을 확인합니다.

19. Joystick_12_AT-Command

▶ 프로그램 상에서 소프트웨어 시리얼 포트를 통해 직접 BLE 장치(FBL780_Serial)에 명령어를 전달합니다..

```
#include <SoftwareSerial.h>

SoftwareSerial bleSerial(A0, A1); // RX, TX
//-----
void setup()
{
  Serial.begin(9600);
  bleSerial.begin(9600);
  Serial.println("AT Command Test");

  delay(500);
  bleSerial.print("at");
  bleSerial.print("\r");
}

void loop()
{
  if(bleSerial.available())
    Serial.write(bleSerial.read());
  if(Serial.available())
    bleSerial.write(Serial.read());
}
```

소프트웨어 시리얼 사용을 위해 "라이브러리"를 추가합니다.

라이브러리를 사용하기 위해 "인자"를 생성합니다.

"Serial.begin(9600);"만 사용하면 하드웨어 시리얼 포트 사용준비가 완료됩니다.

"bleSerial.begin(9600);"만 사용하면 소프트웨어 시리얼 포트 사용준비가 완료됩니다.

하드웨어 시리얼 포트에 출력하고자 하는 시리얼 메시지를 "Serial.println()"에 입력합니다.

여기서는 "AT Command Test"를 입력했습니다.

BLE 장치가 시작되기를 기다리기 위해 약 500ms 대기합니다.

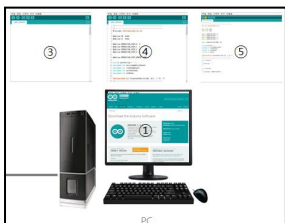
소프트웨어 시리얼 포트에 AT Command를 전달합니다.
전달하고자 하는 AT Command "at"를 bleSerial.print()에 입력합니다.

AT Command 전달이 완료된 것을 알리기 위해 "\r"을 "bleSerial.print()"에 입력합니다.

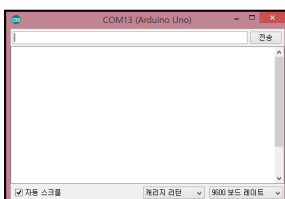
이후부터는 Bypass가 실행됩니다.

"at"부분에 다른 AT Command를 입력해서 테스트를 진행해 봅니다.

Bypass 상태를 이용하여 AT Command 테스트 진행해 봅니다. 특히 드론과 연결을 진행하여 AT Command 입력 후 발생하는 응답 값을 미리 확인하여 다른 예제 진행에 도움이 될 수 있도록 합니다.
(Scan으로 address 확인하는 것도 필요합니다.)



- ① "조이스틱 쉴드"를 아두이노에 장착합니다.
- ② BLE 장치를 "조이스틱 쉴드"에 장착합니다.
- ③ 아두이노 프로그램으로 Joystick_12_AT-Command.ino를 엽니다.
- ④ 아두이노에 다운로드를 진행합니다.



- ⑤ 아두이노 시리얼 창을 실행합니다.
- ⑥ 출력되는 시리얼 데이터 확인합니다.
- ⑦ AT Command 입력 & 응답 값을 확인합니다.

20. Joystick_13_BT-Connect

- ▶ "Switch ~ Case"문과 "PIO Input" 기능을 이용하여 BLE 장치(FBL780_Serial)에 AT Command를 전달하고, 펄테크 드론과 "Bluetooth 연결 / 종료"를 진행합니다.

```
#include <SoftwareSerial.h>

#define OPERATION_STEP_0 0
#define OPERATION_STEP_1 1
#define OPERATION_STEP_2 2

SoftwareSerial bleSerial(A0, A1); // RX, TX
//-----
unsigned int currentStep;
//-----
void setup()
{
    Serial.begin(9600);
    bleSerial.begin(9600);

    for(int i = 5; i < 11; i++)
    {
        pinMode(i, INPUT);
        digitalWrite(i, HIGH);
    }

    currentStep = 0;
}
```

소프트웨어 시리얼 사용을 위해 "라이브러리"를 추가합니다.

"Switch ~ Case"문에 사용할 스텝을 Define문으로 정의합니다.

소프트웨어 시리얼 라이브러리를 사용하기 위해 "인자"를 생성합니다.

"Switch ~ Case"문에 사용할 변수를 정의합니다.

"Serial.begin(9600);"만 사용하면 하드웨어 시리얼 포트 사용준비가 완료됩니다.

"bleSerial.begin(9600);"만 사용하면 소프트웨어 시리얼 포트 사용준비가 완료됩니다.

PIO Port를 Input으로 설정하고 각각의 초기 값을 High로 설정합니다.

currentStep = 0 설정합니다.

```

void loop()
{
  switch(currentStep)
  {
    case OPERATION_STEP_0:
      Serial.println("Connect Test");
      currentStep++;
      break;
    case OPERATION_STEP_1:
      if(!digitalRead(9))
      {
        Serial.println("Pressed Connect Button");
        bleSerial.print("atd");
        bleSerial.print("083a5c1f015b");
        bleSerial.print("\r");
        currentStep++;
      }
      break;
    case OPERATION_STEP_2:
      if(!digitalRead(10))
      {
        Serial.println("Pressed Disconnect Button");
        bleSerial.print("ath");
        bleSerial.print("\r");
        currentStep = OPERATION_STEP_1;
      }
      break;
  }
}

```

"currentStep = 0"에 의해 "case OPERATION_STEP_0:"이 실행됩니다.

"Connect Test" 메시지를 하드웨어 시리얼 포트로 출력합니다.

"currentStep++"을 진행하여 currentStep = 1이 됩니다.

"break"에 의해 "Switch ~ Case"문을 종료합니다.

"currentStep = 1"에 의해 "case OPERATION_STEP_1:"이 실행됩니다.

PIO 9번 포트가 눌렸는지(Low) 체크합니다.

PIO 9번 포트가 눌린 경우,

"Pressed Connect Button" 메시지 출력 /

소프트웨어 시리얼 포트로 "atd083a5c1f015b\r" 출력 / currentStep++ 실행됩니다.

"currentStep++"에 의하여 currentStep = 2가 됩니다.

"break"에 의해 "Switch ~ Case"문을 종료합니다.

"currentStep = 2"에 의해 "case OPERATION_STEP_2:"가 실행됩니다.

PIO 10번 포트가 눌렸는지(Low) 체크합니다.

PIO 10번 포트가 눌린 경우,

"Pressed Disconnect Button" 메시지 출력 /

소프트웨어 시리얼 포트로 "ath\r" 출력 / currentStep = OPERATION_STEP_1 실행됩니다.

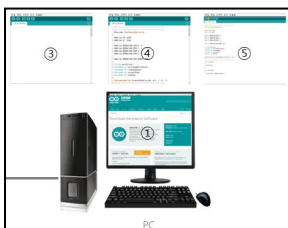
"break"에 의해 "Switch ~ Case"문을 종료합니다.

"case OPERATION_STEP_2:"에서 PIO 10번 포트가 눌리면,

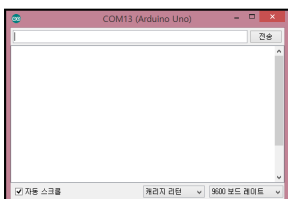
다음에는 "case OPERATION_STEP_1:"이 실행됩니다.

"case OPERATION_STEP_1:"에서 PIO 9번 포트가 눌리면, 다음에는 "case OPERATION_STEP_2:"가 실행됩니다.

※ "atd"명령어 뒤의 숫자(083a5c1f015b)는 연결할 펌테크 드론에 따라 다르게 입력되어야 합니다.



- ① "조이스틱 쉴드"를 아두이노에 장착합니다.
- ② BLE 장치를 "조이스틱 쉴드"에 장착합니다.
(단, BLE는 Central Role 및 SMODE 0로 동작을 해야합니다.)
- ③ 아두이노 프로그램으로 Joystick_13_BT-Connect.ino를 엽니다.
- ④ 아두이노에 다운로드를 진행합니다.



- ⑤ 아두이노 시리얼 창을 실행합니다.
- ⑥ 출력되는 시리얼 데이터 확인합니다.

⑦ 펌테크 드론 전원을 ON 합니다.

⑧ PIO 9번 스위치를 눌러서 블루투스 연결을 진행합니다. (연결되면 "조이스틱 쉴드"의 Status LED가 ON 됩니다.)

⑨ PIO 10번 스위치를 눌러서 블루투스 연결 종료를 진행합니다.

21. Joystick_14_Check-Message_01

- ▶ BLE 장치(FBL780_Serial)에 AT Command 전달 후, 원하는 응답 값 체크를 진행합니다.
"Switch ~ Case"문을 이용하여 순차적으로 진행하면서 AT Command를 전달하고 응답 값을 체크합니다.

```
#include <SoftwareSerial.h>
```

```
#define OPERATION_STEP_0 0
#define OPERATION_STEP_1 1
#define OPERATION_STEP_2 2
#define OPERATION_STEP_3 3
#define OPERATION_STEP_4 4
#define OPERATION_STEP_5 5
#define OPERATION_STEP_6 6
#define OPERATION_STEP_7 7
#define OPERATION_STEP_8 8
#define OPERATION_STEP_9 9
#define OPERATION_STEP_10 10
#define OPERATION_STEP_11 11
#define OPERATION_STEP_12 12
```

```
SoftwareSerial bleSerial(A0, A1); // RX, TX
//-----
String uartString = "";
unsigned int currentStep;
```

소프트웨어 시리얼 사용을 위해 "라이브러리"를 추가합니다.

"Switch ~ Case"문에 사용할 스텝을 Define문으로 정의합니다.

소프트웨어 시리얼 라이브러리를 사용하기 위해 "인자"를 생성합니다.

문자열 저장을 위한 변수를 정의합니다.

"Switch ~ Case"문에 사용할 변수를 정의합니다.

```
void setup()
{
    Serial.begin(9600);
    bleSerial.begin(9600);

    for(int i = 5; i < 11; i++)
    {
        pinMode(i, INPUT);
        digitalWrite(i, HIGH);
    }

    currentStep = 0;
}
```

"Serial.begin(9600);"만 사용하면 하드웨어 시리얼 포트 사용준비가 완료됩니다.

"bleSerial.begin(9600);"만 사용하면 소프트웨어 시리얼 포트 사용준비가 완료됩니다.

PIO Port를 Input으로 설정하고 각각의 초기 값을 High로 설정합니다.

currentStep = 0 설정합니다.

```

void loop()
{
  switch(currentStep)
  {
    case OPERATION_STEP_0:
      Serial.println("Check Message Test 01");
      currentStep++;
      break;
    case OPERATION_STEP_1:
      if(!digitalRead(9))
      {
        Serial.println("Pressed Connect Button");
        bleSerial.print("atd");
        bleSerial.print("083a5c1f015b");
        bleSerial.print("Wr");
        delay(300);
        uartString = "";
        currentStep++;
      }
      break;
    case OPERATION_STEP_2:
      if(bleSerial.available())
      {
        char inChar = bleSerial.read();
        uartString += inChar;
        //
        if( uartString.length() > 4
          && uartString.startsWith("WrWr")
          && uartString.endsWith("WrWr") )
        {
          currentStep++;
        }
      }
      break;
  }
}

```

"currentStep = 0"에 의해 "case OPERATION_STEP_0:"이 실행됩니다.

"Check Message Test 01"메시지를 하드웨어 시리얼 포트에 출력합니다.

"currentStep++"을 진행하여 currentStep = 1이 됩니다.

"break"에 의해 "Switch ~ Case"문을 종료합니다.

"currentStep = 1"에 의해 "case OPERATION_STEP_1:"이 실행됩니다.

PIO 9번 포트가 눌렸는지(Low) 체크합니다.

PIO 9번 포트가 눌린 경우, 하드웨어 시리얼 포트에

"Pressed Connect Button" 메시지 출력 /

소프트웨어 시리얼 포트에 "atd083a5c1f015bWr" 출력 /

currentStep++ 실행됩니다.

300ms 대기후 문자열 저장 변수를 초기화 합니다.

"currentStep++"에 의하여 currentStep = 2가 됩니다.

"break"에 의해 "Switch ~ Case"문을 종료합니다.

"currentStep = 2"에 의해 "case OPERATION_STEP_2:"가 실행됩니다.

소프트웨어 시리얼 포트에 데이터가 입력됐는지 체크합니다.

소프트웨어 시리얼 포트에 데이터가 입력된 경우, 입력된

시리얼 데이터를 읽어서 inChar에 저장합니다. inChar에

저장된 데이터를 문자열 저장 변수 uartString에 추가합니다.

uartString에 저장된 문자열의 길이가 4보다 크고,

"WrWr"으로 시작되며, "WrWr"으로 끝나는지 체크합니다.

(펄테크 제품은 AT Command를 입력한 경우, 모든 응답 값이

"WrWr"으로 시작해서 "WrWr"으로 끝납니다.) 응답 값이

4보다 크고 "WrWr"으로 시작되고 "WrWr"으로 종료되는 경우,

currentStep++를 실행해서 currentStep = 3으로 만듭니다.

"break"에 의해 "Switch ~ Case"문을 종료합니다.

"Wr" = Carriage Return(0x0D) "를 의미합니다.

"Wn" = Line Feed(0x0A) "를 의미합니다.

```

case OPERATION_STEP_3:
    if(uartString.equals("\r\nOK\r\n"))
    {
        Serial.println("Received OK");
        delay(300);
        uartString = "";
        currentStep++;
    }
    break;
case OPERATION_STEP_4:
    if(bleSerial.available())
    {
        char inChar = bleSerial.read();
        uartString += inChar;
        //
        if( uartString.length() > 4
            && uartString.startsWith("\r\n")
            && uartString.endsWith("\r\n") )
        {
            currentStep++;
        }
    }
    break;
case OPERATION_STEP_5:
    if(uartString.startsWith("\r\nCONNECT "))
    {
        Serial.println("Received CONNECT");
        delay(300);
        uartString = "";
        currentStep++;
    }
    break;

```

"currentStep = 3"에 의해 "case OPERATION_STEP_3:"이 실행됩니다.

"equals()"를 사용하여 uartString변수에 저장된 문자열이 "\r\nOK\r\n"인지 비교합니다.

비교 값이 맞는 경우, 하드웨어 시리얼 포트로

"Received OK" 메시지 출력 / 300ms 대기 /

문자열 저장변수 초기화 /

currentStep++ 실행됩니다.

"currentStep++"에 의하여 currentStep = 4가 됩니다.

"break"에 의해 "Switch ~ Case"문을 종료합니다.

"currentStep = 4"에 의해 "case OPERATION_STEP_4:"가 실행됩니다.

소프트웨어 시리얼 포트로 데이터가 입력됐는지 체크합니다.

소프트웨어 시리얼 포트로 데이터가 입력된 경우, 입력된 시리얼 데이터를 읽어서 inChar에 저장합니다. inChar에 저장된 데이터를 문자열 저장 변수 uartString에 추가합니다. uartString에 저장된 문자열의 길이가 4보다 크고, "\r\n"으로 시작되며, "\r\n"으로 끝나는지 체크합니다. (텀테크 제품은 AT Command를 입력한 경우, 모든 응답 값이 "\r\n"으로 시작해서 "\r\n"으로 끝납니다.) 응답 값이 4보다 크고 "\r\n"으로 시작되고 "\r\n"으로 종료되는 경우, currentStep++를 실행해서 currentStep = 5로 만듭니다. "break"에 의해 "Switch ~ Case"문을 종료합니다.

"currentStep = 5"에 의해 "case OPERATION_STEP_5:"가 실행됩니다.

"startsWith()"를 사용하여 uartString변수에 저장된 문자열이 "\r\nCONNECT"로 시작되는지 비교합니다.

비교 값이 맞는 경우, 하드웨어 시리얼 포트로

"Received CONNECT" 메시지 출력 / 300ms 대기 /

문자열 저장변수 초기화 /

currentStep++ 실행됩니다.

"currentStep++"에 의하여 currentStep = 6이 됩니다.

"break"에 의해 "Switch ~ Case"문을 종료합니다.


```

case OPERATION_STEP_6:
    if(bleSerial.available())
    {
        bleSerial.read();
    }

    if(!digitalRead(10))
    {
        Serial.println("Pressed Disconnect Button");
        bleSerial.print("ath");
        bleSerial.print("Wr");
        delay(300);
        uartString = "";
        currentStep++;
    }
break;
case OPERATION_STEP_7:
    if(bleSerial.available())
    {
        char inChar = bleSerial.read();
        uartString += inChar;
        //
        if( uartString.length() > 4
            && uartString.startsWith("WrWr")
            && uartString.endsWith("WrWr") )
        {
            currentStep++;
        }
    }
break;
case OPERATION_STEP_8:
    if(uartString.equals("WrWrOKWrWr"))
    {
        Serial.println("Received OK");
        delay(300);
        uartString = "";
        currentStep++;
    }
break;

```

"currentStep = 6"에 의해 "case OPERATION_STEP_6:"이 실행됩니다.

소프트웨어 시리얼 포트로 데이터가 입력됐는지 체크합니다.

소프트웨어 시리얼 포트로 데이터가 입력된 경우, 입력된 시리얼 데이터를 읽기만 하고 아무런 처리를 하지 않습니다.

PIO 10번 포트가 눌렸는지(Low) 체크합니다.

PIO 10번 포트가 눌린 경우, 하드웨어 시리얼 포트로

"Pressed Disconnect Button" 메시지 출력 /

소프트웨어 시리얼 포트로 "athWr" 출력 / 300ms 대기 /

문자열 저장변수 초기화 /

currentStep++ 실행됩니다.

"currentStep++"에 의하여 currentStep = 7이 됩니다.

"break"에 의해 "Switch ~ Case"문을 종료합니다.

"currentStep = 7"에 의해 "case OPERATION_STEP_7:"이 실행됩니다..

소프트웨어 시리얼 포트로 데이터가 입력됐는지 체크합니다.

소프트웨어 시리얼 포트로 데이터가 입력된 경우, 입력된 시리얼 데이터를 읽어서 inChar에 저장합니다. inChar에 저장된 데이터를 문자열 저장 변수 uartString에 추가합니다.

uartString에 저장된 문자열의 길이가 4보다 크고, "WrWr"으로 시작되며, "WrWr"으로 끝나는지 체크합니다.

(펄테크 제품은 AT Command를 입력한 경우, 모든 응답 값이 "WrWr"으로 시작해서 "WrWr"으로 끝납니다.) 응답 값이 4보다 크고 "WrWr"으로 시작되고 "WrWr"으로 종료되는 경우, currentStep++를 실행해서 currentStep = 8로 만듭니다.

"break"에 의해 "Switch ~ Case"문을 종료합니다.

"currentStep = 8"에 의해 "case OPERATION_STEP_8:"이 실행됩니다.

"equals()"를 사용하여 uartString변수에 저장된 문자열이 "WrWrOKWrWr"인지 비교합니다.

비교 값이 맞는 경우, 하드웨어 시리얼 포트로

"Received OK" 메시지 출력 / 300ms 대기 /

문자열 저장변수 초기화 /

currentStep++ 실행됩니다.

"currentStep++"에 의하여 currentStep = 9가 됩니다.

"break"에 의해 "Switch ~ Case"문을 종료합니다.

```

case OPERATION_STEP_9:
    if(bleSerial.available())
    {
        char inChar = bleSerial.read();
        uartString += inChar;
        //
        if( uartString.length() > 4
            && uartString.startsWith("WrWn")
            && uartString.endsWith("WrWn") )
        {
            currentStep++;
        }
    }
    break;
case OPERATION_STEP_10:
    if(uartString.startsWith("WrWnDISCONNECT"))
    {
        Serial.println("Recieved DISCONNECT");
        delay(300);
        uartString = "";
        currentStep++;
    }
    break;

```

"currentStep = 9"에 의해 "case OPERATION_STEP_9:"가 실행됩니다.

소프트웨어 시리얼 포트로 데이터가 입력됐는지 체크합니다. 소프트웨어 시리얼 포트로 데이터가 입력된 경우, 입력된 시리얼 데이터를 읽어서 inChar에 저장합니다. inChar에 저장된 데이터를 문자열 저장 변수 uartString에 추가합니다. uartString에 저장된 문자열의 길이가 4보다 크고, "WrWn"으로 시작되며, "WrWn"으로 끝나는지 체크합니다. (펄테크 제품은 ATCommand를 입력한 경우 모든 응답 값이 "WrWn"으로 시작해서 "WrWn"으로 끝납니다.) 응답 값이 4보다 크고 "WrWn"으로 시작되고 "WrWn"으로 종료되는 경우, currentStep++를 실행해서 currentStep = 10으로 만듭니다.

"break"에 의해 "Switch ~ Case"문을 종료합니다.

"currentStep = 10"에 의해 "case OPERATION_STEP_10:"이 실행됩니다.

"startsWith()"를 사용하여 uartString변수에 저장된 문자열이 "WrWnDISCONNECT"로 시작되는지 비교합니다.

비교 값이 맞는 경우, 하드웨어 시리얼 포트로 "Received DISCONNECT" 메시지 출력 / 300ms 대기 / 문자열 저장변수 초기화 / currentStep++ 실행됩니다.

"currentStep++"에 의하여 currentStep = 11이 됩니다.

"break"에 의해 "Switch ~ Case"문을 종료합니다.

```

case OPERATION_STEP_11:
    if(bleSerial.available())
    {
        char inChar = bleSerial.read();
        uartString += inChar;
        //
        if( uartString.length() > 4
            && uartString.startsWith("WrWn")
            && uartString.endsWith("WrWn") )
        {
            currentStep++;
        }
    }
    break;
case OPERATION_STEP_12:
    if(uartString.startsWith("WrWnREADY"))
    {
        Serial.println("Recieved READY");
        delay(300);
        uartString = "";
        currentStep = OPERATION_STEP_1;
    }
    break;
}
}

```

"currentStep = 11"에 의해 "case OPERATION_STEP_11:"이 실행됩니다.

소프트웨어 시리얼 포트로 데이터가 입력됐는지 체크합니다.

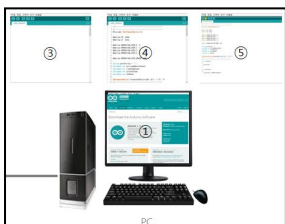
소프트웨어 시리얼 포트로 데이터가 입력된 경우, 입력된 시리얼 데이터를 읽어서 inChar에 저장합니다. inChar에 저장된 데이터를 문자열 저장 변수 uartString에 추가합니다. uartString에 저장된 문자열의 길이가 4보다 크고, "WrWn"으로 시작되며, "WrWn"으로 끝나는지 체크합니다. (펄테크 제품은 AT Command를 입력한 경우, 모든 응답 값이 "WrWn"으로 시작해서 "WrWn"으로 끝납니다.) 응답 값이 4보다 크고 "WrWn"으로 시작되고 "WrWn"으로 종료되는 경우, currentStep++를 실행해서 currentStep = 12로 만듭니다. "break"에 의해 "Switch ~ Case"문을 종료합니다.

"currentStep = 12"에 의해 "case OPERATION_STEP_12:"이 실행됩니다.

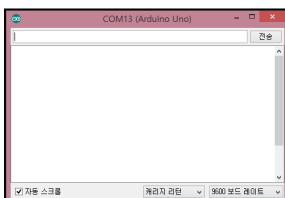
"startsWith()"를 사용하여 uartString변수에 저장된 문자열이 "WrWnREADY"로 시작되는지 비교합니다.

비교 값이 맞는 경우, 하드웨어 시리얼 포트로 "Received READY" 메시지 출력 / 300ms 대기 / 문자열 저장변수 초기화 / currentStep = OPERATION_STEP_1 실행됩니다. "break"에 의해 "Switch ~ Case"문을 종료합니다.

currentStep = OPERATION_STEP_1에 의해 다음 "Switch ~ Case"문 실행 시 "case OPERATION_STEP_1:"이 실행됩니다.



- ① "조이스틱 쉴드"를 아두이노에 장착합니다.
- ② BLE 장치를 "조이스틱 쉴드"에 장착합니다.
- ③ 아두이노 프로그램으로 Joystick_14_Check-Message_01.ino를 엽니다.
- ④ 아두이노에 다운로드를 진행합니다.



- ⑤ 아두이노 시리얼 창을 실행합니다.
- ⑥ 출력되는 시리얼 데이터 확인합니다.

- ⑦ 펄테크 드론 전원을 ON 합니다.
- ⑧ PIO 9번 스위치를 눌러서 블루투스 연결을 진행합니다. (연결되면 "조이스틱 쉴드"의 Status LED가 ON됩니다.)
- ⑨ PIO 10번 스위치를 눌러서 블루투스 연결 종료를 진행합니다.

22. Joystick_14_Check-Message_02

- ▶ Joystick_14_Check-Message_01의 경우, 순차적으로 진행을 하기 때문에 프로그램이 비 효율적으로 운영됩니다.
예를 들어, "소프트웨어 시리얼 포트에 데이터가 입력됐는지 체크"하는 부분이 너무 많이 작성되었습니다.
이 부분을 한 개만 작성하여 프로그램이 조금 더 효율적으로 운영될 수 있도록 합니다.

```
#include <SoftwareSerial.h>

#define OPERATION_STEP_0 0
#define OPERATION_STEP_1 1
#define OPERATION_STEP_2 2
#define OPERATION_STEP_3 3
#define OPERATION_STEP_4 4
#define OPERATION_STEP_5 5
#define OPERATION_STEP_6 6
#define OPERATION_STEP_7 7
#define OPERATION_STEP_8 8

SoftwareSerial bleSerial(A0, A1); // RX, TX
//-----
String uartString = "";
unsigned int currentStep;
unsigned int oldStep;
//-----
void checkNextStep()
{
    delay(300);
    uartString = "";
    oldStep = currentStep;
    currentStep = OPERATION_STEP_0;
}
```

소프트웨어 시리얼 사용을 위해 "라이브러리"를 추가합니다.
"Switch ~ Case"문에 사용할 스텝을 Define문으로 정의합니다.
소프트웨어 시리얼 라이브러리를 사용하기 위해 "인자"를 생성합니다.

문자열 저장을 위한 변수를 정의합니다.
"Switch ~ Case"문에 사용할 변수를 정의합니다.
이전에 수행하던 스텝을 저장할 변수를 정의합니다.

"checkNextStep()" 이름으로 함수를 정의합니다.
이 함수는 300ms 대기 / 문자열 저장변수 초기화 /
oldStep = currentStep 실행 /
currentStep = OPERATION_STEP_0 을 실행합니다.

"checkNextStep()" 함수는 현재 진행 중이던 스텝을 저장하고,
다음 스텝을 OPERATION_STEP_0으로 설정하여 소프트웨어
시리얼 포트에 입력된 데이터를 체크해야 하는 경우에 사용하는
함수입니다.

```
void setup()
{
    Serial.begin(9600);
    bleSerial.begin(9600);

    for(int i = 5; i < 11; i++)
    {
        pinMode(i, INPUT);
        digitalWrite(i, HIGH);
    }

    currentStep = 1;
}
```

"Serial.begin(9600);"만 사용하면 하드웨어 시리얼 포트 사용준비가 완료됩니다.
"bleSerial.begin(9600);"만 사용하면 소프트웨어 시리얼 포트 사용준비가
완료됩니다.

PIO Port를 Input으로 설정하고 각각의 초기 값을 High로 설정합니다.

currentStep = 1 설정합니다.

```

void loop()
{
  switch(currentStep)
  {
    case OPERATION_STEP_0:
      if(bleSerial.available())
      {
        char inChar = bleSerial.read();
        uartString += inChar;
        //
        if( uartString.length() > 4
          && uartString.startsWith("WrWn")
          && uartString.endsWith("WrWn") )
        {
          currentStep = oldStep;
          currentStep++;
        }
      }
      break;
    case OPERATION_STEP_1:
      Serial.println("Check Message Test 02");
      currentStep++;
      break;
    case OPERATION_STEP_2:
      if(!digitalRead(9))
      {
        Serial.println("Pressed Connect Button");
        bleSerial.print("atd");
        bleSerial.print("083a5c1f015b");
        bleSerial.print("Wr");
        checkNextStep();
      }
      break;
    case OPERATION_STEP_3:
      if(uartString.equals("WrWnOKWrWn"))
      {
        Serial.println("Received OK");
        checkNextStep();
      }
      break;
  }
}

```

"case OPERATION_STEP_0:"은 소프트웨어 시리얼 포트에 데이터가 입력됐는지 체크가 필요한 경우 실행되도록 합니다.

소프트웨어 시리얼 포트에 데이터가 입력된 경우, 입력된 시리얼 데이터를 읽어서 inChar에 저장합니다. inChar에 저장된 데이터를 문자열 저장 변수 uartString에 추가합니다. uartString에 저장된 문자열의 길이가 4보다 크고, "WrWn"으로 시작되며, "WrWn"으로 끝나는지 체크합니다.

(펄테크 제품은 AT Command를 입력한 경우, 모든 응답 값이 "WrWn"으로 시작해서 "WrWn"으로 끝납니다.) 응답 값이 4보다 크고 "WrWn" 으로 시작되고 "WrWn"으로 종료되는 경우, currentStep = oldStep을 실행해서 이전에 수행하던 스텝을 복귀하고 currentStep++를 실행해서 이전에 수행하던 다음 스텝으로 만듭니다.

"currentStep = 1"에 의해 "case OPERATION_STEP_1:"이 실행됩니다.

"Check Message Test 02"메시지를 하드웨어 시리얼 포트에 출력합니다.

"currentStep++"를 진행하여 currentStep = 2가 됩니다.

"break"에 의해 "Switch ~ Case"문을 종료합니다.

"currentStep = 2"에 의해 "case OPERATION_STEP_2:"가 실행됩니다.

PIO 9번 포트가 눌렀는지(Low) 체크합니다.

PIO 9번 포트가 눌린 경우, 하드웨어 시리얼 포트에

"Pressed Connect Button" 메시지 출력 /

소프트웨어 시리얼 포트에 "atd083a5c1f015bWr" 출력 /

checkNextStep() 실행됩니다.

"break"에 의해 "Switch ~ Case"문을 종료합니다.

"checkNextStep()" 함수는 현재 진행 중이던 스텝을 저장하고 (oldStep = currentStep), 다음 스텝을 OPERATION_STEP_0으로 설정하여(currentStep = OPERATION_STEP_0)소프트웨어 시리얼 포트에 입력된 데이터를 체크해야 하는 경우에 사용하는 함수입니다.

"checkNextStep()"에 의해 "case OPERATION_STEP_0:"가 실행되는

경우, 소프트웨어 시리얼 포트에 입력된 데이터의 비교가 완료되면

"currentStep = oldStep / currentStep++"에 의해서

"currentStep = 3"이 됩니다.

"currentStep = 3"에 의해 "case OPERATION_STEP_3:"이 실행됩니다.

"equals()"를 사용하여 uartString변수에 저장된 문자열이

"WrWnOKWrWn"인지 비교합니다. 비교 값이 맞는 경우, 하드웨어

시리얼 포트에 "Received OK" 메시지 출력 / checkNextStep() 실행됩니다.

"break"에 의해 "Switch ~ Case"문을 종료합니다.

"checkNextStep()" 함수는 현재 진행 중이던 스텝을 저장하고(oldStep = currentStep), 다음 스텝을 OPERATION_STEP_0으로 설정하여(currentStep = OPERATION_STEP_0) 소프트웨어 시리얼 포트에 입력된 데이터를 체크해야 하는 경우에 사용하는 함수입니다.

"checkNextStep()"에 의해 "case OPERATION_STEP_0:"가 실행되는 경우, 소프트웨어 시리얼 포트에 입력된 데이터의 비교가 완료되면 "currentStep = oldStep / currentStep++"에 의해서 "currentStep = 4"가 됩니다.

```

case OPERATION_STEP_4:
    if(uartString.startsWith("\r\nCONNECT "))
    {
        Serial.println("Received CONNECT");
        delay(300);
        uartString = "";
        currentStep++;
    }
    break;
case OPERATION_STEP_5:
    if(bleSerial.available())
    {
        bleSerial.read();
    }

    if(!digitalRead(10))
    {
        Serial.println("Pressed Disconnect Button");
        bleSerial.print("ath");
        bleSerial.print("\r\n");
        checkNextStep();
    }
    break;
case OPERATION_STEP_6:
    if(uartString.equals("\r\nOK\r\n"))
    {
        Serial.println("Received OK");
        checkNextStep();
    }
    break;

```

"currentStep = 4"에 의해 "case OPERATION_STEP_4:"가 실행됩니다.

"startsWith()"를 사용하여 uartString변수에 저장된 문자열이 "\r\nCONNECT"로 시작되는지 비교합니다.

비교 값이 맞는 경우, 하드웨어 시리얼 포트로

"Received CONNECT" 메시지 출력 / 300ms 대기 / 문자열 저장변수 초기화 / currentStep++ 실행됩니다.

"currentStep++"에 의하여 currentStep = 5가 됩니다.

"break"에 의해 "Switch ~ Case"문을 종료합니다.

"currentStep = 5"에 의해 "case OPERATION_STEP_5:"가 실행됩니다.

소프트웨어 시리얼 포트로 데이터가 입력됐는지 체크합니다.

소프트웨어 시리얼 포트로 데이터가 입력된 경우, 입력된 시리얼 데이터를 읽기만 하고 아무런 처리를 하지 않습니다.

PIO 10번 포트가 눌렀는지(Low) 체크합니다.

PIO 10번 포트가 눌린 경우, 하드웨어 시리얼 포트로

"Pressed Disconnect Button" 메시지 출력 /

소프트웨어 시리얼 포트로 "ath\r\n" 출력 /

checkNextStep() 실행됩니다.

"break"에 의해 "Switch ~ Case"문을 종료합니다.

"checkNextStep()" 함수는 현재 진행 중이던 스텝을 저장하고 (oldStep = currentStep), 다음 스텝을 OPERATION_STEP_0으로 설정하여(currentStep = OPERATION_STEP_0)

소프트웨어 시리얼 포트로 입력된 데이터를 체크해야 하는 경우에 사용하는 함수입니다.

"checkNextStep()"에 의해 "case OPERATION_STEP_0:"가 실행되는 경우, 소프트웨어 시리얼 포트로 입력된 데이터의 비교가 완료되면 "currentStep = oldStep / currentStep++"에 의해서 "currentStep = 6"이 됩니다.

"currentStep = 6"에 의해 "case OPERATION_STEP_6:"이 실행됩니다.

"equals()"를 사용하여 uartString변수에 저장된 문자열이 "\r\nOK\r\n"인지 비교합니다.

비교 값이 맞는 경우, 하드웨어 시리얼 포트로 "Received OK" 메시지 출력 / checkNextStep() 실행됩니다.

"break"에 의해 "Switch ~ Case"문을 종료합니다.

"checkNextStep()" 함수는 현재 진행 중이던 스텝을 저장하고(oldStep = currentStep), 다음 스텝을 OPERATION_STEP_0으로 설정하여(currentStep = OPERATION_STEP_0) 소프트웨어 시리얼 포트로 입력된 데이터를 체크해야 하는 경우에 사용하는 함수입니다.

"checkNextStep()"에 의해 "case OPERATION_STEP_0:"가 실행되는 경우, 소프트웨어 시리얼 포트로 입력된 데이터의 비교가 완료되면 "currentStep = oldStep / currentStep++"에 의해서 "currentStep = 7"이 됩니다.

```

case OPERATION_STEP_7:
    if(uartString.startsWith("#r#nDISCONNECT"))
    {
        Serial.println("Recieved DISCONNECT");
        checkNextStep();
    }
    break;
case OPERATION_STEP_8:
    if(uartString.startsWith("#r#nREADY"))
    {
        Serial.println("Recieved READY");
        delay(300);
        uartString = "";
        currentStep = OPERATION_STEP_2;
    }
    break;
}
}

```

"currentStep = 7"에 의해 "case OPERATION_STEP_7:"이 실행됩니다.

"startsWith()"를 사용하여 uartString변수에 저장된 문자열이 "#r#nDISCONNECT"로 시작되는지 비교합니다.

비교 값이 맞는 경우, 하드웨어 시리얼 포트로

"Received DISCONNECT" 메시지 출력 /

checkNextStep() 실행됩니다.

"break"에 의해 "Switch ~ Case"문을 종료합니다.

"checkNextStep()" 함수는 현재 진행 중이던 스텝을 저장하고 (oldStep = currentStep), 다음 스텝을 OPERATION_STEP_0 으로 설정하여(currentStep = OPERATION_STEP_0) 소프트웨어 시리얼 포트로 입력된 데이터를 체크해야 하는 경우에 사용하는 함수입니다.

"checkNextStep()"에 의해 "case OPERATION_STEP_0:"가 실행되는 경우, 소프트웨어 시리얼 포트로 입력된 데이터의 비교가 완료되면 "currentStep = oldStep / currentStep++"에 의해서 "currentStep = 8"이 됩니다.

"currentStep = 8"에 의해 "case OPERATION_STEP_8:"이 실행됩니다.

"startsWith()"를 사용하여 uartString변수에 저장된 문자열이 "#r#nREADY"로 시작되는지 비교합니다.

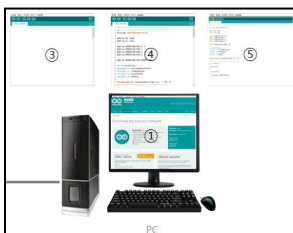
비교 값이 맞는 경우, 하드웨어 시리얼 포트로 "Received READY" 메시지 출력 / 300ms 대기 /

문자열 저장변수 초기화 / currentStep = OPERATION_STEP_2 실행됩니다.

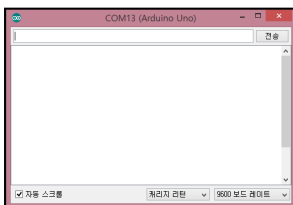
"break"에 의해 "Switch ~ Case"문을 종료합니다.

currentStep = OPERATION_STEP_2에 의해 다음 "Switch ~ Case"문 실행 시

"case OPERATION_STEP_2:"가 실행됩니다.



- ① "조이스틱 쉴드"를 아두이노에 장착합니다.
- ② BLE 장치를 "조이스틱 쉴드"에 장착합니다.
- ③ 아두이노 프로그램으로 Joystick_14_Check-Message_02.ino를 엽니다.
- ④ 아두이노에 다운로드를 진행합니다.



- ④ 아두이노 시리얼 창을 실행합니다.
- ⑤ 출력되는 시리얼 데이터 확인합니다.

⑥ 펄테크 드론의 전원을 ON 합니다.

⑦ PIO 9번 스위치를 눌러서 블루투스 연결을 진행합니다. (연결되면 "조이스틱 쉴드"의 Status LED가 ON됩니다.)

⑧ PIO 10번 스위치를 눌러서 블루투스 연결 종료 진행합니다.

23. Joystick_15_DataPacket_01

▶ 펄테크 드론을 제어하기 위해서 아두이노 "조이스틱 쉴드"에서 송신해야 할 "데이터 패킷"을 테스트 합니다.

BLE Peripheral 장치는 특정한 기능을 제공하기 위해서 Service를 제공합니다.

펄테크 드론의 특정한 기능은 "비행 Service" 입니다.

(참고 1: 펄테크의 FBL780_Serial Peripheral의 특정한 기능은 "시리얼 Service" 입니다.)

(참고 2: 펄테크의 FBL780_H Peripheral의 특정한 기능은 "시리얼 Service / PIO Service / ADC Service / Config Service" 입니다.)

BLE Peripheral 장치의 특정한 기능을 제어(데이터 송신/수신)하기 위해서는 "핸들"을 사용합니다.

(펄테크 홈페이지에서 FBL780BC_H 제품의 "FBL780_Appendix_1~3.pdf" 문서를 참조바랍니다.)

펄테크 드론의 "비행 Service"를 제어하기 위해서는 "0x0006"의 "핸들 번호"를 사용합니다. 즉 핸들번호 "0006"에 드론 제어를 위해 정해진 프로토콜을 송신합니다.

BLE Peripheral 장치를 제어하기 위해서는 BLE Central 장치를 사용해야 합니다.

펄테크 드론을 제어하기 위해서는 펄테크 BLE 장치인 FBL780_Serial을 Central로 설정하여 사용합니다.

FBL780_Serial을 Central로 설정하고 아두이노 "조이스틱 쉴드"에 장착하여 펄테크 드론 제어를 위한 "데이터 패킷"을 송신합니다. (Central 설정은 "18. Joystick_11_Serial_02"에서 진행했습니다.)

펄테크 드론 제어를 위한 프로토콜의 "데이터 패킷" 구조는 아래와 같습니다. (총 8바이트 사용)

		좌,우 이동	전진,후진	좌,우 회전	상승,하강		
startBit	commandBit	roll	pitch	yaw	throttle	operationBit	checkSum
F0	A1	0~200	0~200	0~200	0~200	5	0

-startBit / commandBit : 고정 값 (0xF0, 0xA1)

-roll : 100을 기준으로 100보다 크면 오른쪽으로 이동하고, 100보다 작으면 왼쪽으로 이동합니다. (0~200)

-pitch : 100을 기준으로 100보다 크면 전진하고, 100보다 작으면 후진합니다. (0~200)

-yaw : 100을 기준으로 100보다 크면 우회전하고, 100보다 작으면 좌회전합니다. (0~200)

-throttle : 0에서 시작해서 증가하면 상승하고, 감소하면 하강합니다. (0~200)

-operationBit : 1(높이 수동 조정) / 5(높이 자동 조정)

-checkSum : 송신 데이터의 오류 검출에 사용합니다.

$checkSum = commandBit + roll + pitch + yaw + throttle + operationBit$



※ "높이 수동 조정"의 경우, 반드시 Safety Track에서 테스트를 진행하시기 바랍니다.

※ "높이 수동 조정"의 경우 드론 제어가 힘들기 때문에 드론 파손의 위험이 많습니다.

```
#include <SoftwareSerial.h>

SoftwareSerial bleSerial(A0, A1); // RX, TX
//-----
void setup()
{
  Serial.begin(9600);
  bleSerial.begin(9600);
  Serial.println("Data Packet Test 01");

  for(int i = 5; i < 11; i++)
  {
    pinMode(i, INPUT);
    digitalWrite(i, HIGH);
  }

  delay(500);

  bleSerial.print("atd");
  bleSerial.print("083a5c1f015b");
  bleSerial.print("\r");

  delay(500);
}
```

소프트웨어 시리얼 사용을 위해 "라이브러리"를 추가합니다.
소프트웨어 시리얼 라이브러리를 사용하기 위해 "인자"를 생성합니다.

"Serial.begin(9600);"만 사용하면 하드웨어 시리얼 포트 사용준비가 완료됩니다.
"bleSerial.begin(9600);"만 사용하면 소프트웨어 시리얼 포트 사용준비가 완료됩니다.
하드웨어 시리얼 포트에 출력하고자 하는 시리얼 메시지를 "Serial.println()"에 입력합니다.
여기서는 "Data Packet Test 01"을 입력했습니다.

PIO Port를 Input으로 설정하고 초기 값을 각각 High로 설정합니다.

BLE 장치가 시작되기를 기다리기 위해 약 500ms 대기합니다.

펄테크 드론과 연결하기 위해 소프트웨어 시리얼 포트에 "atd083a5c1f015b\r"를 출력합니다.

펄테크 드론과 연결되기를 기다리기 위해 약 500ms 대기합니다.

잠시 후, 펄테크 드론과 연결되면 "조이스틱 쉴드"의 Status LED가 ON 됩니다.

```
void loop()
{
  if(!digitalRead(5))
  {
    bleSerial.print("at+writeh0006");
    //
    bleSerial.print("f0");
    bleSerial.print("a1");
    bleSerial.print("64");
    bleSerial.print("64");
    bleSerial.print("64");
    bleSerial.print("78");
    bleSerial.print("01");
    bleSerial.print("46");
    //
    bleSerial.print("\r");
    delay(300);
  }
}
```

PIO 5번 포트가 눌렸는지(Low) 체크합니다.

PIO 5번 포트가 눌린 경우, 소프트웨어 시리얼 포트에 "at+writeh0006"을 출력합니다. 이것은 펄테크 드론의 0006 핸들에 HEX로 데이터를 전달하겠다는 AT Command입니다.

소프트웨어 시리얼 포트에 "f0"를 출력합니다.. 이것은 펄테크 드론으로 startBit를 송신하는 것입니다.

소프트웨어 시리얼 포트에 "a1"을 출력합니다.. 이것은 펄테크 드론으로 commandBit를 송신하는 것입니다.

소프트웨어 시리얼 포트에 "64"를 출력합니다. 이것은 펄테크 드론으로 roll을 송신하는 것입니다. (0x64 = 100으로 좌/우 이동 안 함)

소프트웨어 시리얼 포트에 "64"를 출력합니다. 이것은 펄테크 드론으로 pitch를 송신하는 것입니다. (0x64 = 100으로 전/후 이동 안 함)

소프트웨어 시리얼 포트에 "64"를 출력합니다. 이것은 펄테크 드론으로 yaw를 송신하는 것입니다.

(0x64 = 100으로 좌회전/우회전 안 함)

소프트웨어 시리얼 포트로 "78"를 출력합니다. 이것은 펄테크 드론으로 throttle을 송신하는 것입니다.

(0x78 = 120으로 120의 세기로 상승 하라는 뜻입니다.)

소프트웨어 시리얼 포트로 "01"을 출력합니다. 이것은 펄테크 드론으로 "높이 수동 조정"을 송신하는 것입니다.

소프트웨어 시리얼 포트로 "46"을 출력합니다. 이것은 펄테크 드론으로 checksum을 송신하는 것입니다.

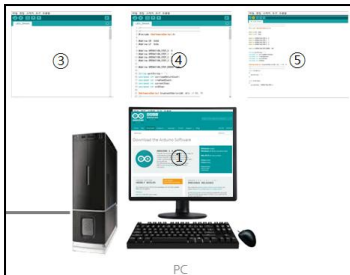
(checksum = 0xa1 + 0x64 + 0x64 + 0x64 + 0x78 + 0x01 : 14. Joystick_08_Checksum_02 참고)

소프트웨어 시리얼 포트로 "Wr"을 출력합니다. 이것은 데이터의 입력이 완료되었다는 AT Command입니다.

약 300ms 시간 동안 대기합니다.

PIO 5번 포트가 눌러있는 동안 "조이스틱 쉴드"를 통해 펄테크 드론으로 제어를 위한 프로토콜이 약 300ms 간격으로 계속 송신됩니다.

PIO 5번 포트의 누름을 해제하면 펄테크 드론은 바로 하강합니다.



- ① "조이스틱 쉴드"를 아두이노에 장착합니다.
- ② BLE 장치를 "조이스틱 쉴드"에 장착합니다.
- ③ 아두이노 프로그램으로 Joystick_15_DataPacket_01.ino를 엽니다.
- ④ 아두이노에 다운로드를 진행합니다.

⑤ USB 케이블 제거해서 아두이노 전원을 OFF 합니다.

⑥ 펄테크 드론의 전원을 ON 합니다.

⑦ 아두이노 전원을 ON 합니다.

⑧ 잠시 후 블루투스가 연결됩니다. (연결되면 "조이스틱 쉴드"의 Status LED가 ON 됩니다.)

⑨ PIO 5번 스위치를 눌러서 펄테크 드론 제어를 위한 프로토콜을 송신합니다.

⑩ 펄테크 드론이 상승합니다. 단, PIO 5번 스위치 지속적으로 누름 상태여야 합니다.

⑪ PIO 5번 스위치 누름 해제 시 펄테크 드론은 하강합니다.

24. Joystick_15_DataPacket_02

- ▶ 이전에 테스트했던 결과에 의하면, 펌테크 드론을 제어하기 위해서는 "데이터 패킷"을 일정한 간격으로 지속적으로 송신해야 합니다. (이전에는 "PIO 5" 포트를 누르고 있는 동안 "데이터 패킷"이 지속적으로 송신되었습니다.)

이번에는 펌테크 드론과 연결된 이후 일정한 간격으로 "데이터 패킷"이 지속적으로 송신되도록 합니다.
그리고 throttle(상승/하강) 제어를 함수로 만들고 "상승/하강"이 제어되도록 합니다.

throttle(상승/하강)값이 변경되면 checksum도 변경되므로, checksum 계산도 함수를 이용하여 계산하도록 합니다.

```
#include <SoftwareSerial.h>

SoftwareSerial bleSerial(A0, A1); // RX, TX
//-----
unsigned char startBit = 0xf0;
unsigned char commandBit = 0xa1;
unsigned char roll = 100;
unsigned char pitch = 100;
unsigned char yaw = 100;
unsigned char throttle = 0;
unsigned char operationBit = 0x01;
unsigned char checksum = 0;
//-----

void checkThrottle()
{
    //throttle: 감소시 하강, 증가시 상승
    if(!digitalRead(6))
    {
        if(throttle > 59)
            throttle -= 20;
        else if(throttle > 3)
            throttle -= 4;
    }
    else if(!digitalRead(5))
    {
        if(throttle < 20)
            throttle = 20;
        else if(throttle < 181)
            throttle += 20;
    }
}

void checkCRC()
{
    checksum = commandBit + roll + pitch + yaw + throttle + operationBit;
    checksum = checksum & 0x00ff;
}
```

소프트웨어 시리얼 사용을 위해 "라이브러리"를 추가합니다.

소프트웨어 시리얼 라이브러리를 사용하기 위해 "인자"를 생성합니다.

펌테크 드론 제어를 위한 "데이터 패킷"을 변수로 정의합니다.

"checkThrottle()" 이름으로 함수를 정의합니다.

이 함수는 "PIO 5" 포트와 "PIO 6" 포트의 눌림을 체크합니다.

"PIO 6" 포트가 눌린 경우, throttle(상승/하강) 값을 감소시킵니다.

"PIO 5" 포트가 눌린 경우, throttle(상승/하강) 값을 증가시킵니다.

"checkCRC()" 이름으로 함수를 정의합니다.

이 함수는 펌테크 드론 제어를 위한 "데이터 패킷"의 checksum을 계산합니다.

checksum 계산은 "데이터 패킷"으로 사용하는 변수를 이용합니다.

```

void setup()
{
    Serial.begin(9600);
    bleSerial.begin(9600);
    Serial.println("Data Packet Test 02");

    for(int i = 5; i < 11; i++)
    {
        pinMode(i, INPUT);
        digitalWrite(i, HIGH);
    }

    delay(500);

    bleSerial.print("atd");
    bleSerial.print("083a5c1f015b");
    bleSerial.print("#r");

    delay(500);
}

```

"Serial.begin(9600);"만 사용하면 하드웨어 시리얼 포트 사용준비가 완료됩니다.

"bleSerial.begin(9600);"만 사용하면 소프트웨어 시리얼 포트 사용준비가 완료됩니다.

하드웨어 시리얼 포트에 출력하고자 하는 시리얼 메시지를 "Serial.println()"에 입력합니다.

여기서는 "Data Packet Test 02"를 입력했습니다.

PIO Port를 Input으로 설정하고 각각의 초기 값을 High로 설정합니다.

BLE 장치가 시작되기를 기다리기 위해 약 500ms 대기합니다.

펌테크 드론과 연결하기 위해 소프트웨어 시리얼 포트에

"atd083a5c1f015bWr"를 출력합니다.

펌테크 드론과 연결되기를 기다리기 위해 약 500ms 대기합니다.

펌테크 드론과 연결되면 "조이스틱 쉴드"의 Status LED가 ON 됩니다.

```

void loop()
{
    checkThrottle();
    //
    checkCRC();
    //
    bleSerial.print("at+writeh0006");
    //
    bleSerial.print("f0");
    bleSerial.print("a1");
    bleSerial.print("64");
    bleSerial.print("64");
    bleSerial.print("64");
    //
    if(throttle < 0x10)
        bleSerial.print("0" + String(throttle, HEX));
    else
        bleSerial.print(String(throttle, HEX));
    //
    bleSerial.print("01");
    //
    if(checkSum < 0x10)
        bleSerial.print("0" + String(checkSum, HEX));
    else
        bleSerial.print(String(checkSum, HEX));
    //
    bleSerial.print("#r");
    delay(100);
}

```

"checkThrottle()" 함수를 콜해서 throttle 값을 계산합니다.

"checkCRC()" 함수를 콜해서 checkSum 값을 계산합니다.

소프트웨어 시리얼 포트에 "at+writeh0006"을 출력합니다.

이것은 펌테크 드론의 0006 핸들에 HEX로 데이터를 전달하겠다는 AT Command입니다.

소프트웨어 시리얼 포트에 "f0"를 출력합니다. 이것은 펌테크 드론으로 startBit를 송신하는 것입니다.

소프트웨어 시리얼 포트에 "a1"을 출력합니다. 이것은 펌테크 드론으로 commandBit를 송신하는 것입니다.

소프트웨어 시리얼 포트에 "64"를 출력합니다. 이것은 펌테크 드론으로 roll을 송신하는 것입니다.
(0x64 = 100으로 좌/우 이동 안 함)

소프트웨어 시리얼 포트에 "64"를 출력합니다.. 이것은 펌테크 드론으로 pitch를 송신하는 것입니다.
(0x64 = 100으로 전/후 이동 안 함)

소프트웨어 시리얼 포트에 "64"를 출력합니다. 이것은 펌테크 드론으로 yaw를 송신하는 것입니다.
(0x64 = 100으로 좌회전/우회전 안 함)

소프트웨어 시리얼 포트로 throttle을 출력합니다. 이것은 펄테크 드론으로 throttle을 송신하는 것입니다.

소프트웨어 시리얼 포트로 "01"을 출력합니다. 이것은 펄테크 드론으로 "높이 수동 조정"을 송신하는 것입니다.

소프트웨어 시리얼 포트로 checksum을 출력합니다. 이것은 펄테크 드론으로 checksum을 송신하는 것입니다.
(checksum = commandBit + roll + pitch + yaw + throttle + operationBit)

소프트웨어 시리얼 포트로 "Wr"을 출력합니다. 이것은 데이터의 입력이 완료되었다는 AT Command입니다.

약 100ms 시간 동안 대기합니다.

※ print(String(throttle, HEX)) / print(String(throttle))의 차이점

print(String(throttle, HEX)) → 실제 값(숫자)을 HEX 타입의 문자로 변경하여 출력합니다.

print(String(throttle)) → 실제 값을 10진수 타입의 문자로 변경하여 출력합니다.

Hex와 10진수는 10진수 10의 값부터 표시 방법이 달라집니다.

"Joystick_15_DataPacket_02-1.ino"를 다운로드 하여 차이점을 확인합니다.

※ print("0" + String(throttle)) / print(String(throttle))의 차이점

16진수 0 ~ f 까지의 값은 한 자리수로 인식이 됩니다. 이것을 문자로 변경해도 한 자리로 인식됩니다.

16진수를 두 자리수로 처리하기 위해 0 ~ f 까지의 값인 경우 print("0" + String(throttle, HEX))를 사용하여 인위적으로 문자 "0"을 추가해 줍니다.

예) at+writeh0006F0A1010203040506

0x0006 = 펄테크 드론의 "비행 Service"를 제어하기 위해서는 "0x0006"의 "핸들 번호"를 사용합니다.

F0A1... = 펄테크 드론 제어를 위한 프로토콜의 "데이터 패킷" 입니다. (총 8바이트 사용)

F0(startBit), A1(commandBit), 01(roll), 02(pitch), 03(yaw), 04(throttle), 05(operationBit),
06(checksum)을 나타냅니다.

bleSerial.print(String(throttle,Hex)) 로 throttle 값을 보낼 경우, throttle 값이 0x0F 보다 작을 시 Hex 값에서 0을 제외하고 4의 값만 전송하여 정확한 데이터 패킷이 아니기 때문에 드론 제어가 불가능합니다.

정상 데이터 패킷 : at+writeh0006F0A1010203040506

잘못된 데이터 패킷 : at+writeh0006F0A101020340506

위의 문제를 해결하기 위하여 bleSerial.print("0" + String(throttle, HEX)) 형태 String 형 변수를 이용하여 throttle 값을 문자로 만들고 만들어진 문자 앞에 0을 붙여 Hex 타입에 문자로 만들어 04의 값을 전송합니다.

"Joystick_15_DataPacket_02-2.ino"를 다운로드 하여 차이점을 확인합니다.



- ① "조이스틱 쉘드"를 아두이노에 장착합니다.
- ② BLE 장치를 "조이스틱 쉘드"에 장착합니다.
- ③ 아두이노 프로그램으로 Joystick_15_DataPacket_02.ino를 엽니다.
- ④ 아두이노에 다운로드를 진행합니다.

- ⑤ USB 케이블을 제거하고 아두이노 전원을 OFF 합니다.
- ⑥ 펄테크 드론의 전원을 ON 합니다.
- ⑦ 아두이노 전원을 ON 합니다.
- ⑧ 잠시 후 블루투스가 연결됩니다. (연결되면 "조이스틱 쉘드"의 Status LED가 ON됩니다.)
- ⑨ PIO 5번 스위치를 눌러서 throttle 팩킷 데이터를 송신합니다.
- ⑩ 펄테크 드론이 상승합니다.
- ⑪ PIO 6번 스위치를 눌러서 throttle 팩킷 데이터를 송신합니다.
- ⑫ 펄테크 드론이 하강합니다.

25. Joystick_15_DataPacket_03

▶ bleSerial.print("f0")나 bleSerial.print("64")의 경우, 변경이 불가능한 문자를 출력하는 것입니다.

조금 더 유용한 프로그램 작성을 위해 변경이 불가능한 문자를 출력하지 말고 변경이 가능한 변수에 저장된 숫자를 문자로 변경하여 출력하는 방식을 사용합니다.

```
#include <SoftwareSerial.h>

SoftwareSerial bleSerial(A0, A1); // RX, TX
//-----
unsigned char startBit = 0xf0;
unsigned char commandBit = 0xa1;
unsigned char roll = 100;
unsigned char pitch = 100;
unsigned char yaw = 100;
unsigned char throttle = 0;
unsigned char operationBit = 0x01;
unsigned char checkSum = 0;
//-----

void checkThrottle()
{
    //throttle: 감소시 하강, 증가시 상승
    if(!digitalRead(6))
    {
        if(throttle > 59)
            throttle -= 20;
        else if(throttle > 3)
            throttle -= 4;
    }
    else if(!digitalRead(5))
    {
        if(throttle < 20)
            throttle = 20;
        else if(throttle < 181)
            throttle += 20;
    }
}

void checkCRC()
{
    checkSum = commandBit + roll + pitch + yaw + throttle + operationBit;
    checkSum = checkSum & 0x00ff;
}
```

소프트웨어 시리얼 사용을 위해 "라이브러리"를 추가합니다.

소프트웨어 시리얼 라이브러리를 사용하기 위해 "인자"를 생성합니다.

펄테크 드론 제어를 위한 "데이터 패킷"을 변수로 정의합니다.

"checkThrottle()" 이름으로 함수를 정의합니다.

이 함수는 "PIO 5" 포트와 "PIO 6" 포트의 눌림을 체크합니다.

"PIO 6" 포트가 눌린 경우, throttle(상승/하강) 값을 감소시킵니다.

"PIO 5" 포트가 눌린 경우, throttle(상승/하강) 값을 증가시킵니다.

"checkCRC()" 이름으로 함수를 정의합니다.

이 함수는 펄테크 드론 제어를 위한 "데이터 패킷"의 checkSum을 계산합니다.

checkSum 계산은 "데이터 패킷"으로 사용하는 변수를 이용합니다.


```

void setup()
{
  Serial.begin(9600);
  bleSerial.begin(9600);
  Serial.println("Data Packet Test 03");

  for(int i = 5; i < 11; i++)
  {
    pinMode(i, INPUT);
    digitalWrite(i, HIGH);
  }

  delay(500);

  bleSerial.print("atd");
  bleSerial.print("083a5c1f015b");
  bleSerial.print("\r");

  delay(500);
}

```

"Serial.begin(9600);"만 사용하면 하드웨어 시리얼 포트 사용준비가 완료됩니다.

"bleSerial.begin(9600);"만 사용하면 소프트웨어 시리얼 포트 사용준비가 완료됩니다.

하드웨어 시리얼 포트에 출력하고자 하는 시리얼 메시지를 "Serial.println()"에 입력합니다.

여기서는 "Data Packet Test 03"을 입력했습니다.

PIO Port를 Input으로 설정하고 각각의 초기 값을 High로 설정합니다.

BLE 장치가 시작되기를 기다리기 위해 약 500ms 대기합니다.

펄테크 드론과 연결하기 위해 소프트웨어 시리얼 포트에 "atd083a5c1f015b\r"를 출력합니다.

펄테크 드론과 연결되기를 기다리기 위해 약 500ms 대기합니다.

펄테크 드론과 연결되면 "조이스틱 쉴드"의 Status LED가 ON 됩니다.

```

void loop()
{
  checkThrottle();
  //
  checkCRC();
  //
  bleSerial.print("at+writeh0006");
  //
  bleSerial.print(String(startBit, HEX));
  bleSerial.print(String(commandBit, HEX));
  bleSerial.print(String(roll, HEX));
  bleSerial.print(String(pitch, HEX));
  bleSerial.print(String(yaw, HEX));
  //
  if(throttle < 0x10)
    bleSerial.print("0" + String(throttle, HEX));
  else
    bleSerial.print(String(throttle, HEX));
  //
  bleSerial.print("0" + String(operationBit, HEX));
  //
  if(checkSum < 0x10)
    bleSerial.print("0" + String(checkSum, HEX));
  else
    bleSerial.print(String(checkSum, HEX));
  //
  bleSerial.print("\r");
  delay(100);
}

```

"checkThrottle()" 함수를 콜해서 throttle 값을 계산합니다.

"checkCRC()" 함수를 콜해서 checkSum 값을 계산합니다.

소프트웨어 시리얼 포트에 "at+writeh0006"을 출력합니다. 이것은 펄테크 드론의 0006 핸들에 HEX로 데이터를 전달하겠다는 AT Command입니다.

소프트웨어 시리얼 포트에 startBit를 출력합니다. 이것은 펄테크 드론으로 startBit를 송신하는 것입니다.

소프트웨어 시리얼 포트에 commandBit를 출력합니다. 이것은 펄테크 드론으로 commandBit를 송신하는 것입니다.

소프트웨어 시리얼 포트에 roll을 출력합니다. 이것은 펄테크 드론으로 roll을 송신하는 것입니다.

소프트웨어 시리얼 포트에 pitch를 출력합니다. 이것은 펄테크 드론으로 pitch를 송신하는 것입니다.

소프트웨어 시리얼 포트에 yaw를 출력합니다. 이것은 펄테크 드론으로 yaw를 송신하는 것입니다.

소프트웨어 시리얼 포트에 throttle을 출력합니다. 이것은 펄테크 드론으로 throttle을 송신하는 것입니다.

소프트웨어 시리얼 포트로 operationBit를 출력합니다.

이것은 펌테크 드론으로 "높이 수동 조정"을 송신하는 것입니다.

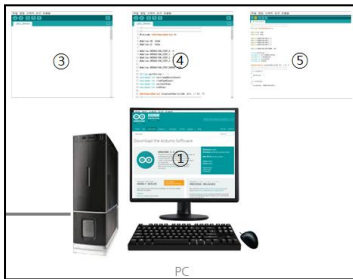
소프트웨어 시리얼 포트로 checkSum을 출력합니다. 이것은 펌테크 드론으로 checkSum을 송신하는 것입니다.
(checkSum = commandBit + roll + pitch + yaw + throttle + operationBit)

소프트웨어 시리얼 포트로 "Wr"을 출력합니다. 이것은 데이터의 입력이 완료되었다는 AT Command입니다.

약 100ms 시간 동안 대기합니다.

※ roll / pitch / yaw의 경우, 아직은 변하지 않고 100(0x64)를 출력하기 때문에 문자 "0"을 출력하는 부분이 추가되지 않았습니다.

※ operationBit의 경우, 1(0x01)을 출력하기 때문에 문자 "0"을 조건 없이 출력하도록 합니다.



- ① "조이스틱 쉴드"를 아두이노에 장착합니다.
- ② BLE 장치를 "조이스틱 쉴드"에 장착합니다.
- ③ 아두이노 프로그램으로 Joystick_15_DataPacket_03.ino를 엽니다.
- ④ 아두이노에 다운로드를 진행합니다.

⑤ USB 케이블을 제거하고 아두이노 전원을 OFF 합니다.

⑥ 펌테크 드론의 전원을 ON 합니다.

⑦ 아두이노 전원을 ON 합니다.

⑧ 잠시 후 블루투스가 연결됩니다. (연결되면 "조이스틱 쉴드"의 Status LED가 ON됩니다.)

⑨ PIO 5번 스위치를 눌러서 throttle 패킷 데이터를 송신합니다.

⑩ 펌테크 드론이 상승합니다.

⑪ PIO 6번 스위치를 눌러서 throttle 패킷 데이터를 송신합니다.

⑫ 펌테크 드론이 하강합니다.

26. Joystick_15_DataPacket_04

▶ 이번에는 펄테크 드론 pitch(전진/후진) 제어를 함수로 만들고 "전진/후진"이 제어되도록 합니다.

```
#include <SoftwareSerial.h>

SoftwareSerial bleSerial(A0, A1); // RX, TX
//-----
unsigned char startBit = 0xf0;
unsigned char commandBit = 0xa1;
unsigned char roll = 100;
unsigned char pitch = 100;
unsigned char yaw = 100;
unsigned char throttle = 0;
unsigned char operationBit = 0x01;
unsigned char checksum = 0;
//-----
void checkThrottle()
{
    //throttle: 감소시 하강, 증가시 상승
    if(!digitalRead(6))
    {
        if(throttle > 59)
            throttle -= 20;
        else if(throttle > 3)
            throttle -= 4;
    }
    else if(!digitalRead(5))
    {
        if(throttle < 20)
            throttle = 20;
        else if(throttle < 181)
            throttle += 20;
    }
}
```

소프트웨어 시리얼 사용을 위해 "라이브러리"를 추가합니다.

소프트웨어 시리얼 라이브러리를 사용하기 위해 "인자"를 생성합니다.

펄테크 드론 제어를 위한 "데이터 패킷"을 변수로 정의합니다.

"checkThrottle()" 이름으로 함수를 정의합니다.

이 함수는 "PIO 5" 포트와 "PIO 6" 포트의 높임을 체크합니다.

"PIO 6" 포트가 눌린 경우, throttle(상승/하강) 값을 감소시킵니다.

"PIO 5" 포트가 눌린 경우, throttle(상승/하강) 값을 증가시킵니다.

```
void checkPitch()
{
    //pitch: 증가시 전진, 감소시 후진
    unsigned int secondPitch = analogRead(5);

    if(secondPitch < 400)
        pitch = 50;
    else if(secondPitch > 600)
        pitch = 150;
    else
        pitch = 100;
}

void checkCRC()
{
    checksum = commandBit + roll + pitch + yaw + throttle + operationBit;
    checksum = checksum & 0x00ff;
}
```

"checkPitch()" 이름으로 함수를 정의합니다.

이 함수는 "아날로그 5"포트를 읽습니다.

"아날로그 5"포트가 400보다 작은 경우, pitch (전진/후진) 값을 50으로 설정합니다.

"아날로그 5"포트가 600보다 큰 경우, pitch (전진/후진) 값을 150으로 설정합니다.

"아날로그 5"포트가 400 ~ 600 사이인 경우, pitch(전진/후진) 값을 100으로 설정합니다.

"checkCRC()" 이름으로 함수를 정의합니다.

이 함수는 펄테크 드론 제어를 위한 "데이터 패킷"의 checksum을 계산합니다. checksum 계산은 "데이터 패킷"으로 사용하는 변수를 이용합니다.

```

void setup()
{
  Serial.begin(9600);
  bleSerial.begin(9600);
  Serial.println("Data Packet Test 04");

  for(int i = 5; i < 11; i++)
  {
    pinMode(i, INPUT);
    digitalWrite(i, HIGH);
  }

  delay(500);

  bleSerial.print("atd");
  bleSerial.print("083a5c1f015b");
  bleSerial.print("\r");

  delay(500);
}

```

"Serial.begin(9600);"만 사용하면 하드웨어 시리얼 포트 사용준비가 완료됩니다.

"bleSerial.begin(9600);"만 사용하면 소프트웨어 시리얼 포트 사용준비가 완료됩니다.

하드웨어 시리얼 포트에 출력하고자 하는 시리얼 메시지를 "Serial.println()"에 입력합니다.

여기서는 "Data Packet Test 04"를 입력했습니다.

PIO Port를 Input으로 설정하고 각각의 초기 값을 High로 설정합니다.

BLE 장치가 시작되기를 기다리기 위해 약 500ms 대기합니다.

펄테크 드론과 연결하기 위해 소프트웨어 시리얼 포트에 "atd083a5c1f015b\r"를 출력합니다.

펄테크 드론과 연결되기를 기다리기 위해 약 500ms 대기합니다.

펄테크 드론과 연결되면 "조이스틱 쉼드"의 Status LED가 ON 됩니다.

```

void loop()
{
  checkThrottle();
  checkPitch();
  //
  checkCRC();
  //
  bleSerial.print("at+writeh0006");
  //
  bleSerial.print(String(startBit, HEX));
  bleSerial.print(String(commandBit, HEX));
  bleSerial.print(String(roll, HEX));
  bleSerial.print(String(pitch, HEX));
  bleSerial.print(String(yaw, HEX));
  //
  if(throttle < 0x10)
    bleSerial.print("0" + String(throttle, HEX));
  else
    bleSerial.print(String(throttle, HEX));
  //
  bleSerial.print("0" + String(operationBit, HEX));
  //
  if(checkSum < 0x10)
    bleSerial.print("0" + String(checkSum, HEX));
  else
    bleSerial.print(String(checkSum, HEX));
  //
  bleSerial.print("\r");
  delay(100);
}

```

"checkThrottle()" 함수를 콜해서 throttle 값을 계산합니다.

"checkPitch()" 함수를 콜해서 pitch 값을 계산합니다.

"checkCRC()" 함수를 콜해서 checkSum 값을 계산합니다.

소프트웨어 시리얼 포트에 "at+writeh0006"을 출력합니다.

이것은 펄테크 드론의 0006 핸들에 HEX로 데이터를 전달하겠다는 AT Command입니다.

소프트웨어 시리얼 포트에 startBit를 출력합니다. 이것은 펄테크 드론으로 startBit를 송신하는 것입니다.

소프트웨어 시리얼 포트에 commandBit를 출력합니다. 이것은 펄테크 드론으로 commandBit를 송신하는 것입니다.

소프트웨어 시리얼 포트에 roll을 출력합니다. 이것은 펄테크 드론으로 roll을 송신하는 것입니다.

소프트웨어 시리얼 포트에 pitch를 출력합니다. 이것은 펄테크 드론으로 pitch를 송신하는 것입니다.

소프트웨어 시리얼 포트에 yaw를 출력합니다. 이것은 펄테크 드론으로 yaw를 송신하는 것입니다.

소프트웨어 시리얼 포트에 throttle을 출력합니다. 이것은 펄테크 드론으로 throttle을 송신하는 것입니다.

소프트웨어 시리얼 포트로 operationBit를 출력합니다. 이것은 펄테크 드론으로 "높이 수동 조정"을 송신하는 것입니다.

소프트웨어 시리얼 포트로 checkSum을 출력합니다. 이것은 펄테크 드론으로 checkSum을 송신하는 것입니다. (checkSum = commandBit + roll + pitch + yaw + throttle + operationBit)

소프트웨어 시리얼 포트로 "Wr"을 출력합니다. 이것은 데이터의 입력이 완료되었다는 AT Command입니다.

약 100ms 시간 동안 대기합니다.

※ pitch의 경우, 50(0x32) / 100(0x64) / 150(0x96)값만 가지도록 했기 때문에 문자 "0"을 출력하는 부분이 추가되지 않았습니다.

※ roll / yaw의 경우, 아직은 변하지 않고 100(0x64)를 출력하기 때문에 문자 "0"을 출력하는 부분이 추가되지 않았습니다.

※ operationBit의 경우, 1(0x01)을 출력하기 때문에 문자 "0"을 조건 없이 출력하도록 합니다.



- ① "조이스틱 쉴드"를 아두이노에 장착합니다.
- ② BLE 장치를 "조이스틱 쉴드"에 장착합니다.
- ③ 아두이노 프로그램으로 Joystick_15_DataPacket_04.ino를 엽니다.
- ④ 아두이노에 다운로드를 진행합니다.

⑤ USB 케이블을 제거하고 아두이노 전원을 OFF 합니다.

⑥ 펄테크 드론의 전원을 ON 합니다.

⑦ 아두이노 전원을 ON합니다.

⑧ 잠시 후 블루투스가 연결됩니다. (연결되면 "조이스틱 쉴드"의 Status LED가 ON 됩니다.)

⑨ PIO 5번 스위치를 눌러서 throttle 패킷 데이터를 송신합니다.

⑩ 펄테크 드론이 상승합니다.

⑪ ADC 5번 포트를 읽어서(조이스틱 앞으로 조정) pitch 패킷 데이터를 송신합니다.

⑫ 펄테크 드론이 전진합니다. (앞으로 기울어짐)

⑬ ADC 5번 포트를 읽어서(조이스틱 뒤로 조정) pitch 패킷 데이터를 송신합니다.

⑭ 펄테크 드론이 후진합니다. (뒤로 기울어짐)

⑮ ADC 5번 포트를 읽어서(조이스틱 중앙으로 조정) pitch 패킷 데이터를 송신합니다.

⑯ 펄테크 드론이 수평이 됩니다. (기울어짐 없음)

⑰ PIO 6번 스위치를 눌러서 throttle 패킷 데이터를 송신합니다.

⑱ 펄테크 드론이 하강합니다.

27. Joystick_15_DataPacket_05

▶ 이번에는 펄테크 드론 roll(좌이동/우이동) 제어를 함수로 만들고 "좌이동/우이동"이 제어되도록 합니다.

```
#include <SoftwareSerial.h>

SoftwareSerial bleSerial(A0, A1); // RX, TX

//-----
unsigned char startBit = 0xf0;
unsigned char commandBit = 0xa1;
unsigned char roll = 100;
unsigned char pitch = 100;
unsigned char yaw = 100;
unsigned char throttle = 0;
unsigned char operationBit = 0x01;
unsigned char checksum = 0;
//-----

void checkThrottle()
{
    //throttle: 감소시 하강, 증가시 상승
    if(!digitalRead(6))
    {
        if(throttle > 59)
            throttle -= 20;
        else if(throttle > 3)
            throttle -= 4;
    }
    else if(!digitalRead(5))
    {
        if(throttle < 20)
            throttle = 20;
        else if(throttle < 181)
            throttle += 20;
    }
}
```

소프트웨어 시리얼 사용을 위해 "라이브러리"를 추가합니다.

소프트웨어 시리얼 라이브러리를 사용하기 위해 "인자"를 생성합니다.

펄테크 드론 제어를 위한 "데이터 패킷"을 변수로 정의합니다.

"checkThrottle()" 이름으로 함수를 정의합니다.

이 함수는 "PIO 5" 포트와 "PIO 6" 포트의 놀림을 체크합니다.

"PIO 6" 포트가 눌린 경우, throttle(상승/하강) 값을 감소시킵니다.

"PIO 5" 포트가 눌린 경우, throttle(상승/하강) 값을 증가시킵니다.

```
void checkPitch()
{
    //pitch: 증가시 전진, 감소시 후진
    unsigned int secondPitch = analogRead(5);

    if(secondPitch < 400)
        pitch = 50;
    else if(secondPitch > 600)
        pitch = 150;
    else
        pitch = 100;
}

void checkRoll()
{
    //roll: 증가시 오른쪽 미동, 감소시 왼쪽 미동
    unsigned int secondRoll = analogRead(4);

    if(secondRoll < 400)
        roll = 50;
    else if(secondRoll > 600)
        roll = 150;
    else
        roll = 100;
}

void checkCRC()
{
    checksum = commandBit + roll + pitch + yaw + throttle + operationBit;
    checksum = checksum & 0x00ff;
}
```

"checkPitch()" 이름으로 함수를 정의합니다.

이 함수는 "아날로그 5"포트를 읽습니다.

"아날로그 5"포트가 400보다 작은 경우, pitch(전진/후진) 값을 50으로 설정합니다.

"아날로그 5"포트가 600보다 큰 경우, pitch(전진/후진) 값을 150으로 설정합니다.

"아날로그 5"포트가 400 ~ 600 사이인 경우, pitch(전진/후진) 값을 100으로 설정합니다.

"checkRoll()" 이름으로 함수를 정의합니다.

이 함수는 "아날로그 4"포트를 읽습니다.

"아날로그 4"포트가 400보다 작은 경우, roll(좌이동/우이동) 값을 50으로 설정합니다.

"아날로그 4"포트가 600보다 큰 경우, roll(좌이동/우이동) 값을 150으로 설정합니다.

"아날로그 4"포트가 400 ~ 600 사이인 경우, roll(좌이동/우이동) 값을 100으로 설정합니다.

"checkCRC()" 이름으로 함수를 정의합니다.

이 함수는 펄테크 드론 제어를 위한 "데이터 패킷"의 checksum을 계산합니다. checksum 계산은 "데이터 패킷"으로 사용하는 변수를 이용합니다.

```

void setup()
{
  Serial.begin(9600);
  bleSerial.begin(9600);
  Serial.println("Data Packet Test 05");

  for(int i = 5; i < 11; i++)
  {
    pinMode(i, INPUT);
    digitalWrite(i, HIGH);
  }

  delay(500);

  bleSerial.print("atd");
  bleSerial.print("083a5c1f015b");
  bleSerial.print("¶r");

  delay(500);
}

```

"Serial.begin(9600);"만 사용하면 하드웨어 시리얼 포트 사용준비가 완료됩니다.
 "bleSerial.begin(9600);"만 사용하면 소프트웨어 시리얼 포트 사용준비가 완료됩니다.
 하드웨어 시리얼 포트에 출력하고자 하는 시리얼 메시지를 "Serial.println()"에 입력합니다.
 여기서는 "Data Packet Test 05"를 입력했습니다.

PIO Port를 Input으로 설정하고 각각의 초기 값을 High로 설정합니다.

BLE 장치가 시작되기를 기다리기 위해 약 500ms 대기합니다.

펄테크 드론과 연결하기 위해 소프트웨어 시리얼 포트에 "atd083a5c1f015b¶r"를 출력합니다.

펄테크 드론과 연결되기를 기다리기 위해 약 500ms 대기합니다.
 펄테크 드론과 연결되면 "조이스틱 쉼드"의 Status LED가 ON 됩니다.

```

void loop()
{
  checkThrottle();
  checkPitch();
  checkRoll();
  //
  checkCRC();
  //
  bleSerial.print("at+writeh0006");
  //
  bleSerial.print(String(startBit, HEX));
  bleSerial.print(String(commandBit, HEX));
  bleSerial.print(String(roll, HEX));
  bleSerial.print(String(pitch, HEX));
  bleSerial.print(String(yaw, HEX));
  //
  if(throttle < 0x10)
    bleSerial.print("0" + String(throttle, HEX));
  else
    bleSerial.print(String(throttle, HEX));
  //
  bleSerial.print("0" + String(operationBit, HEX));
  //
  if(checkSum < 0x10)
    bleSerial.print("0" + String(checkSum, HEX));
  else
    bleSerial.print(String(checkSum, HEX));
  //
  bleSerial.print("¶r");
  delay(100);
}

```

"checkThrottle()" 함수를 콜해서 throttle 값을 계산합니다.
 "checkPitch()" 함수를 콜해서 pitch 값을 계산합니다.
 "checkRoll()" 함수를 콜해서 roll 값을 계산합니다.
 "checkCRC()" 함수를 콜해서 checkSum 값을 계산합니다.

소프트웨어 시리얼 포트에 "at+writeh0006"을 출력합니다.
 이것은 펄테크 드론의 0006 핸들에 HEX로 데이터를 전달하겠다는 AT Command입니다.

소프트웨어 시리얼 포트에 startBit를 출력합니다. 이것은 펄테크 드론으로 startBit를 송신하는 것입니다.

소프트웨어 시리얼 포트에 commandBit를 출력합니다. 이것은 펄테크 드론으로 commandBit를 송신하는 것입니다.

소프트웨어 시리얼 포트에 roll을 출력합니다. 이것은 펄테크 드론으로 roll을 송신하는 것입니다.

소프트웨어 시리얼 포트에 pitch를 출력합니다. 이것은 펄테크 드론으로 pitch를 송신하는 것입니다.

소프트웨어 시리얼 포트에 yaw를 출력합니다. 이것은 펄테크 드론으로 yaw를 송신하는 것입니다.

소프트웨어 시리얼 포트에 throttle을 출력합니다. 이것은 펄테크 드론으로 throttle을 송신하는 것입니다.

소프트웨어 시리얼 포트로 operationBit를 출력합니다. 이것은 펄테크 드론으로 "높이 수동 조정"을 송신하는 것입니다.

소프트웨어 시리얼 포트로 checkSum을 출력합니다. 이것은 펄테크 드론으로 checkSum을 송신하는 것입니다. (checkSum = commandBit + roll + pitch + yaw + throttle + operationBit)

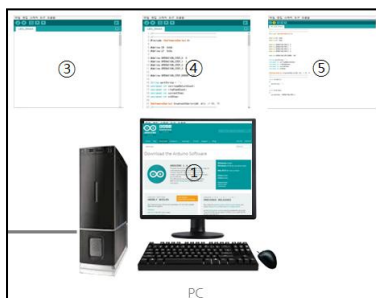
소프트웨어 시리얼 포트로 "Wr"을 출력합니다. 이것은 데이터의 입력이 완료되었다는 AT Command입니다.

약 100ms 시간 동안 대기합니다.

※ pitch / roll의 경우, 50(0x32) / 100(0x64) / 150(0x96)값만 가지도록 했기 때문에 문자 "0"을 출력하는 부분이 추가되지 않았습니다.

※ yaw의 경우, 아직은 변하지 않고 100(0x64)를 출력하기 때문에 문자 "0"을 출력하는 부분이 추가되지 않았습니다.

※ operationBit의 경우, 1(0x01)을 출력하기 때문에 문자 "0"을 조건 없이 출력하도록 합니다.



- ① "조이스틱 쉴드"를 아두이노에 장착합니다.
- ② BLE 장치를 "조이스틱 쉴드"에 장착합니다.
- ③ 아두이노 프로그램으로 Joystick_15_DataPacket_05.ino를 엽니다.
- ④ 아두이노에 다운로드를 진행합니다.

⑤ USB 케이블 제거해서 아두이노 전원을 OFF 합니다.

⑥ 펄테크 드론 전원을 ON합니다.

⑦ 아두이노 전원을 ON 합니다.

⑧ 잠시 후 블루투스가 연결됩니다. (연결되면 "조이스틱 쉴드"의 Status LED가 ON 됩니다.)

⑨ PIO 5번 스위치를 눌러서 throttle 패킷 데이터를 송신합니다.

⑩ 펄테크 드론이 상승합니다.

⑪ ADC 4번 포트를 읽어서(조이스틱 오른쪽으로 조정) roll 패킷 데이터를 송신합니다.

⑫ 펄테크 드론이 우측으로 이동합니다. (우측으로 기울어짐)

⑬ ADC 4번 포트를 읽어서(조이스틱 왼쪽으로 조정) roll 패킷 데이터를 송신합니다.

⑭ 펄테크 드론이 좌측으로 이동합니다. (좌측으로 기울어짐)

⑮ ADC 4번 포트를 읽어서(조이스틱 중앙으로 조정) roll 패킷 데이터를 송신합니다.

⑯ 펄테크 드론이 수평이 됩니다. (기울어짐 없음)

⑰ PIO 6번 스위치를 눌러서 throttle 패킷 데이터를 송신합니다.

⑱ 펄테크 드론이 하강합니다.

28. Joystick_15_DataPacket_06

▶ 이번에는 펄테크 드론 yaw(좌회전/우회전) 제어를 함수로 만들고 "좌회전/우회전"이 제어되도록 합니다.

※ yaw(좌회전/우회전)는 "Safety Track"에서는 테스트가 불가능합니다.

비상버튼을 체크해서 모터 회전을 0으로 설정하도록 합니다.

"데이터 패킷" 송신 부분을 함수로 구성합니다.

```
#include <SoftwareSerial.h>

SoftwareSerial bleSerial(A0, A1); // RX, TX
//-----
unsigned char startBit = 0xf0;
unsigned char commandBit = 0xa1;
unsigned char roll = 100;
unsigned char pitch = 100;
unsigned char yaw = 100;
unsigned char throttle = 0;
unsigned char operationBit = 0x01;
unsigned char checkSum = 0;
//-----
void checkThrottle()
{
    //throttle: 감소시 하강, 증가시 상승
    if(!digitalRead(6))
    {
        if(throttle > 59)
            throttle -= 20;
        else if(throttle > 3)
            throttle -= 4;
    }
    else if(!digitalRead(5))
    {
        if(throttle < 20)
            throttle = 20;
        else if(throttle < 181)
            throttle += 20;
    }
}
```

소프트웨어 시리얼 사용을 위해 "라이브러리"를 추가합니다.

소프트웨어 시리얼 라이브러리를 사용하기 위해 "인자"를 생성합니다.

펄테크 드론 제어를 위한 "데이터 패킷"을 변수로 정의합니다.

"checkThrottle()" 이름으로 함수를 정의합니다.

이 함수는 "PIO 5" 포트와 "PIO 6" 포트의 눌림을 체크합니다.

"PIO 6" 포트가 눌린 경우, throttle(상승/하강) 값을 감소시킵니다.

"PIO 5" 포트가 눌린 경우, throttle(상승/하강) 값을 증가시킵니다.

```

void checkPitch()
{
    //pitch: 증가시 전진, 감소시 후진
    unsigned int secondPitch = analogRead(5);

    if(secondPitch < 400)
        pitch = 50;
    else if(secondPitch > 600)
        pitch = 150;
    else
        pitch = 100;
}

void checkRoll()
{
    //roll: 증가시 오른쪽 이동, 감소시 왼쪽 이동
    unsigned int secondRoll = analogRead(4);

    if(secondRoll < 400)
        roll = 50;
    else if(secondRoll > 600)
        roll = 150;
    else
        roll = 100;
}

void checkYaw()
{
    //Yaw: 감소시 좌회전, 증가시 우회전
    if(!digitalRead(7))
        yaw = 50;
    else if(!digitalRead(8))
        yaw = 150;
    else
        yaw = 100;
}

```

"checkPitch()" 이름으로 함수를 정의합니다.

이 함수는 "아날로그 5"포트를 읽습니다.

"아날로그 5"포트가 400보다 작은 경우, pitch(전진/후진) 값을 50으로 설정합니다.

"아날로그 5"포트가 600보다 큰 경우, pitch(전진/후진) 값을 150으로 설정합니다.

"아날로그 5"포트가 400 ~ 600 사이인 경우, pitch(전진/후진) 값을 100으로 설정합니다.

"checkRoll()" 이름으로 함수를 정의합니다.

이 함수는 "아날로그 4"포트를 읽습니다.

"아날로그 4"포트가 400보다 작은 경우, roll(좌이동/우이동) 값을 50으로 설정합니다.

"아날로그 4"포트가 600보다 큰 경우, roll(좌이동/우이동) 값을 150으로 설정합니다.

"아날로그 4"포트가 400 ~ 600 사이인 경우, roll(좌이동/우이동) 값을 100으로 설정합니다.

"checkYaw()" 이름으로 함수를 정의합니다.

이 함수는 "PIO 7/8"포트를 읽습니다.

"PIO 7"포트가 눌린 경우, yaw(좌회전/우회전) 값을 50으로 설정합니다.

"PIO 8"포트가 눌린 경우, yaw(좌회전/우회전) 값을 150으로 설정합니다.

"PIO 7/8"포트가 모두 눌리지 않은 경우, yaw(좌회전/우회전) 값을 100으로 설정합니다.

```

void checkEmergency()
{
    //비상버튼 눌리면, 모터 회전 즉시 0으로 설정.
    if(!digitalRead(9))
    {
        throttle = 0;
        roll = 100;
        pitch = 100;
        yaw = 100;
    }
}

void sendDroneCommand()
{
    bleSerial.print("at+writeh0006");
    //
    bleSerial.print(String(startBit,HEX));
    bleSerial.print(String(commandBit,HEX));
    bleSerial.print(String(roll,HEX));
    bleSerial.print(String(pitch,HEX));
    bleSerial.print(String(yaw,HEX));
    //
    if(throttle < 0x10)
        bleSerial.print("0" + String(throttle,HEX));
    else
        bleSerial.print(String(throttle,HEX));
    //
    bleSerial.print("0" + String(operationBit,HEX));
    //
    if(checkSum < 0x10)
        bleSerial.print("0" + String(checkSum,HEX));
    else
        bleSerial.print(String(checkSum,HEX));
    //
    bleSerial.print("Wr");
}

```

"checkEmergency()" 이름으로 함수를 정의합니다.
이 함수는 "PIO 9"포트를 읽습니다.

"PIO 9"포트가 눌린 경우, throttle(상승/하강) 값을 0 /
roll(좌이동/우이동) 값을 100 / pitch(전진/후진) 값을 100 /
yaw(좌회전/우회전) 값을 100으로 설정합니다.

"sendDroneCommand()" 이름으로 함수를 정의합니다.
이 함수는 펄테크 드론 제어 데이터 패킷을 송신합니다.

소프트웨어 시리얼 포트에 "at+writeh0006"을 출력합니다.
이것은 펄테크 드론의 0006 핸들에 HEX로 데이터를
전달하겠다는 AT Command입니다.

소프트웨어 시리얼 포트에 startBit를 출력합니다. 이것은 펄테크
드론으로 startBit를 송신하는 것입니다.

소프트웨어 시리얼 포트에 commandBit를 출력합니다. 이것은
펄테크 드론으로 commandBit를 송신하는 것입니다.

소프트웨어 시리얼 포트에 roll을 출력합니다. 이것은 펄테크
드론으로 roll을 송신하는 것입니다.

소프트웨어 시리얼 포트에 pitch를 출력합니다. 이것은 펄테크
드론으로 pitch를 송신하는 것입니다.

소프트웨어 시리얼 포트에 yaw를 출력합니다. 이것은 펄테크
드론으로 yaw를 송신하는 것입니다.

소프트웨어 시리얼 포트에 throttle을 출력합니다. 이것은 펄테크
드론으로 throttle을 송신하는 것입니다.

소프트웨어 시리얼 포트에 operationBit를 출력합니다. 이것은
펄테크 드론으로 "높이 수동 조정"을 송신하는 것입니다.

소프트웨어 시리얼 포트에 checkSum을 출력합니다. 이것은
펄테크 드론으로 checkSum을 송신하는 것입니다.

(checkSum = commandBit + roll + pitch + yaw + throttle + operationBit)

소프트웨어 시리얼 포트에 "Wr"을 출력합니다. 이것은 데이터의 입력이 완료되었다는 AT Command입니다.

```

void checkCRC()
{
    checkSum = commandBit + roll + pitch + yaw + throttle + operationBit;
    checkSum = checkSum & 0x00ff;
}

```

"checkCRC()" 이름으로 함수를 정의합니다.

이 함수는 펄테크 드론 제어를 위한
"데이터 패킷"의 checkSum을 계산합니다.

checkSum 계산은 "데이터 패킷"으로 사용하는 변수를 이용합니다.

```

void setup()
{
  Serial.begin(9600);
  bleSerial.begin(9600);
  Serial.println("Data Packet Test 06");

  for(int i = 5; i < 11; i++)
  {
    pinMode(i, INPUT);
    digitalWrite(i, HIGH);
  }

  delay(500);

  bleSerial.print("atd");
  bleSerial.print("083a5c1f015b");
  bleSerial.print("Wr");

  delay(500);
}

void loop()
{
  checkThrottle();
  checkPitch();
  checkRoll();
  checkYaw();
  checkEmergency();
  //
  checkCRC();
  //
  sendDroneCommand();
  delay(100);
}

```

"Serial.begin(9600);"만 사용하면 하드웨어 시리얼 포트 사용준비가 완료됩니다.

"bleSerial.begin(9600);"만 사용하면 소프트웨어 시리얼 포트 사용준비가 완료됩니다.

하드웨어 시리얼 포트에 출력하고자 하는 시리얼 메시지를 "Serial.println()"에 입력합니다.

여기서는 "Data Packet Test 06"을 입력했습니다.

PIO Port를 Input으로 설정하고 각각의 초기 값을 High로 설정합니다.

BLE 장치가 시작되기를 기다리기 위해 약 500ms 대기합니다.

펄테크 드론과 연결하기 위해 소프트웨어 시리얼 포트에 "atd083a5c1f015bWr"를 출력합니다.

펄테크 드론과 연결되기를 기다리기 위해 약 500ms 대기합니다.

펄테크 드론과 연결되면 "조이스틱 쉴드"의 Status LED가 ON 됩니다.

"checkThrottle()" 함수를 콜해서 throttle 값을 계산합니다.

"checkPitch()" 함수를 콜해서 pitch 값을 계산합니다.

"checkRoll()" 함수를 콜해서 roll 값을 계산합니다.

"checkYaw()" 함수를 콜해서 yaw 값을 계산합니다.

"checkEmergency()" 함수를 콜해서 비상상태를 체크합니다.

"checkCRC()" 함수를 콜해서 checksum 값을 계산합니다.

"sendDroneCommnd()" 함수를 콜해서 펄테크 드론 제어용 "데이터 패킷"을 송신합니다.

약 100ms 시간 동안 대기합니다.

※ pitch / roll / yaw의 경우, 50(0x32) / 100(0x64) / 150(0x96)값만 가지도록 했기 때문에 문자 "0"을 출력하는 부분이 추가되지 않았습니다.

※ operationBit의 경우, 1(0x01)을 출력하기 때문에 문자 "0"을 조건 없이 출력하도록 합니다.



- ① "조이스틱 쉴드"를 아두이노에 장착합니다.
- ② BLE 장치를 "조이스틱 쉴드"에 장착합니다.
- ③ 아두이노 프로그램으로 Joystick_15_DataPacket_06.ino를 엽니다.
- ④ 아두이노에 다운로드를 진행합니다.

⑤ USB 케이블 제거해서 아두이노 전원을 OFF 합니다.

⑥ 펄테크 드론의 전원을 ON 합니다.

⑦ 아두이노 전원을 ON 합니다.

⑧ 잠시 후 블루투스가 연결됩니다. (연결되면 "조이스틱 쉴드"의 Status LED가 ON 됩니다.)

⑨ PIO 5번 스위치를 눌러서 throttle 패킷 데이터를 송신합니다.

⑩ 펄테크 드론이 상승합니다.

⑪ PIO 7번 스위치를 눌러서 yaw 패킷 데이터를 송신합니다.

⑫ 펄테크 드론이 좌회전 합니다. (Safety Track에서 확인이 불가능 합니다.)

⑬ PIO 8번 스위치를 눌러서 yaw 패킷 데이터를 송신합니다.

⑭ 펄테크 드론이 우회전 합니다. (Safety Track에서 확인이 불가능 합니다.)

⑮ PIO 6번 스위치를 눌러서 throttle 패킷 데이터를 송신합니다.

⑯ 펄테크 드론이 하강합니다.

29. Joystick_15_DataPacket_07

▶ 펄테크 드론을 "높이 자동 조정" 모드로 동작시킵니다.

※ "높이 자동 조정" 모드는 "Safety Track" 없이 펄테크 드론을 제어합니다.

※ "높이 자동 조정" 모드는 펄테크 드론의 높이를 임의로 조정할 수 없습니다.

```
#include <SoftwareSerial.h>

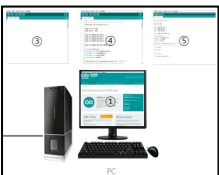
SoftwareSerial bleSerial(A0, A1); // RX, TX
//-----
unsigned char startBit = 0xf0;
unsigned char commandBit = 0xa1;
unsigned char roll = 100;
unsigned char pitch = 100;
unsigned char yaw = 100;
unsigned char throttle = 0;
unsigned char operationBit = 0x05;
unsigned char checksum = 0;
//-----
```

"높이 자동 조정" 모드로 동작시키는 경우, operationBit를 0x05로 변경만 하면 됩니다.

```
void loop()
{
  checkThrottle();
  checkPitch();
  checkRoll();
  checkYaw();
  checkEmergency();
  //
  checkCRC();
  //
  sendDroneCommand();
  delay(10);
}
```

※ "높이 자동 조정" 모드 동작 시, 펄테크 드론이 상승 했다가 바로 떨어지는 경우는 "패킷 데이터" 송신 간격을 짧게 해야 합니다.
(delay(100)을 delay(10)으로 조정)

※ "높이 자동 조정" 모드 동작 시, 펄테크 드론을 상승 시켰을 때 프로펠러가 느리게 돌면, 비상버튼(PIO 9 포트)을 길게 눌러 "데이터 패킷"을 초기화 한 후 다시 상승 시킵니다.



- ① "조이스틱 쉴드"를 아두이노에 장착합니다.
- ② BLE 장치를 "조이스틱 쉴드"에 장착합니다.
- ③ 아두이노 프로그램으로 Joystick_15_DataPacket_07.ino를 엽니다.
- ④ 아두이노에 다운로드를 진행합니다.

⑤ USB 케이블 제거해서 아두이노 전원을 OFF 합니다.

⑥ 펄테크 드론 전원을 ON 합니다.

⑦ 아두이노 전원을 ON 합니다.

⑧ 잠시 후 블루투스가 연결됩니다. (연결되면 "조이스틱 쉴드"의 Status LED가 ON 됩니다.)

⑨ PIO 5번 스위치를 눌러서 throttle 패킷 데이터를 송신합니다.

⑩ 펄테크 드론이 상승합니다. (자동으로 높이가 조정됩니다. → Safety Track 끝까지 올라갑니다.)

⑪ PIO 6번 스위치를 눌러서 throttle 패킷 데이터를 송신합니다.

⑫ 펄테크 드론이 하강합니다.

30. Joystick_20_Drone_Shield

- ▶ "높이 자동 조정"모드로 변경하고, "Switch ~ Case문"을 이용하여 "PIO 버튼 체크"를 통한 블루투스 연결 / 종료, 블루투스 연결 완료 "응답 메시지 체크", "PIO 버튼과 ADC(조이스틱) 입력"을 체크하여 펄테크 드론을 제어 하도록 구성하고 프로그램 효율을 위한 "함수"를 구성하면 펄테크 드론 제어용 "최종 조이스틱 쉴드 프로그램"이 완성됩니다.

최종 소스 코드 설명은 펄테크 홈페이지의 드론 항목의 "Arduino+Drone Kit(Quick Guide)" 문서를 참조하시기 바랍니다.