**[Serial Receiver Provider] SPEKTRUM1024 & SPEKTRUM2048 - Spektrum Remote Receiver Interfacing**

| | | | |
|---|---|---|---|
| **노트북:** | Interface | | |
| **만든 날짜:** | 2017-05-16 오전 8:46 | **수정한 날짜:** | 2017-07-07 오후 3:05 |
| **작성자:** | duvallee.lee@gmail.com | | |

---

### DSM
Spektrum사는 RC radio 시스템에서 사용하던 AM, FM, PCM 방식에서 새로운 DSM(Digital Spectrum Modulation) 방식의 Modulation을 2004년도에 release하였다.
DSM은 "Direct Sequencing Spread Spectrum"을 기반으로 optimiz된 version이다. DSM은 4096 bit의 resolution을 갖고 5.6 ms의 응답 시간을 갖는다.

**DSM SYSTEM SPECIFICATIONS**
. Frequency Band 2.400~2.4835GHz (83.5 MHz)
. Channels 79
. Channel Spacing 1MHz
. Range 3000 ft (914.4 m, 0.9 km)
. Latency 5.6 ms
. Resolution/Channel 4096 steps

### HOW DSM WORKS
"Direct Sequencing Spread Spectrum System"은 같은 주파수로 같은 공간에서 충돌이 발생하기 때문에 충돌을 회피해야 한다.
DSM은 79개의 채널을 지원하고 사용하지 않는 채널을 검색하여 회피하도록 한다.

**Transmitter**
power-on시에 2.4GHz Band에 있는 사용 가능한 79개의 채널을 모두 searching한다. Transmitter는 oepn된 채널을 찾으면 해당 채널을 lock하고 사용가능하지 않더라도 기다리지 않고 다음 채널을 계속 searching하여 79 전체 채널을 검색한다. 보통 79개의 채널을 검색하는데 약 2초 정도가 걸린다. 한 번 검색된 채널은 Transmitter의 power가 off가 되기 전까지 유지를 한다.
Transmitter는 공장에서 GUID(Globally Unique Identification code)로 serial code가 write되고, 해당 serial code에 의해서만 transmitter는 응답을 한다.

**Receiver**
Receiver가 power-on되면 2.4GHz Band에서 transmitter의 인코드된 serial code를 통해 transmitter를 찾는다. 만약 찾으면 해당 채널을 lock한다. 만약 RF signal을 읽어버리면, Receiver는 hold mode로 진입을 하고 RF signal을 다시 획득할 때까지 fail-safe에 저장된 값으로 servo 값을 유지한다.
만약 transmitter가 power-on되기 전에 receiver가 먼저 power-on이 되면 transmitter의 signal을 찾을 때까지 계속 채널을 검색한다. 채널을 검색하는 동안은 Receiver는 fail-safe에 저장된 값으로 유지된다.

### BINDING
Transmitter의 Serial code(GUID)를 Receiver에 프로그램을 하는 것입니다. Receiver의 push-button을 약 30초간 누르면 됩니다.

### TELEMETRY TO COME
Spektrum은 양방향 통신을 지원한다. RC 카의 엔진이나 모터의 온도, 속도, RPM, 배터리 voltage와 같은 정보를 receiver에 전달할 수 있다.

### ETC
Spektrum사는 DSM이후에 DSM2 (2nd generation of Spektrum DSM technology), DSMX을 발표하였다.

### SPEKTRUM1024
1024 resolution을 갖는 PROTOCOL로써 Spektrum의 DSM2 22ms 방식에서만 사용한다.

### SPEKTRUM2048
2048 resolution을 갖는 PROTOCOL로써 Spektrum의 DSM2 22ms를 제외하고는 모두 2048을 사용한다.

### Internal Mode
Spektrum의 internal mode로 동작되는 Receiver는 오직 한개만 transmitter에 연결될 수 있고, 16 bytes Packet data에 system type이 포함된다.

### External Mode
Spektrum의 external mode로 동작되는 Receiver의 여러개를 한 개의 transmitter에 연결될 수 있고, 16 bytes Packet data에 system type은 포함되지 않고 fade counter만 헤더에 있다..

---

# Hardware-Level Protocol

11ms 나 22ms 마다 16-byte data packet를 전송한다.
UART 전송 속도
. BaudRate  : 125000 bps or 115200 bps
. Data bits : 8 bits
. Parity    : No Parity
. Stop bits : 1 stop

### Binding
Receiver가 power-up 시에 transmitter를 찾은 후 연결하는 과정이다.

- Cleanflight에서도 지원하고 있으며, 우리 project에서는 사용하지 않는다.

### Bind pin

transmiter는 bind pin을 통해 입력 받은 pulse의 갯수를 통해 Mode와 Protocol type을 결정한다.
transmitter는 power-on 후에 200ms 동안 pulse의 갯수을 입력 받는다. pulse는 한 주기는 약 240 us이다.

**DSMX Bind Modes : (Recommended)**

| Pulses | Mode | Protocol Type |
|---|---|---|
| ***7*** | ***Internal*** | ***DSMx 22ms*** |
| 8 | External | DSMx 22ms |
| ***9*** | ***Internal*** | ***DSMx 11ms*** |
| 10 | External | DSMx 11ms |

**DSM2 Bind Modes : Not recommended**

| Pulses | Mode | Protocol Type | |
|---|---|---|---|
| 3 | Internal | DSM2 22ms | - only 1024 packet mode |
| 4 | External | DSM2 22ms | - only 1024 packet mode |
| 5 | Internal | DSM2 11ms | |
| 6 | External | DSM2 11ms | |

### Internal and External Modes

Receivers는 master이거나 auxiliary로 연결될 수 있다. 대부분의 리시버는 Master(internal mode)로 동작되며, transmitter(조정기)와는 bind mode를 통해서 연결을 완료 할 수 있다. 이러한 이유로 하나의 리시버에는 Master mode로 동작되는 transmitter(조정기)는 하나만 연결되어야 한다. 그 외 연결되는 transmitter는 auxiliary(External mode)로 연결되어야 한다.

---

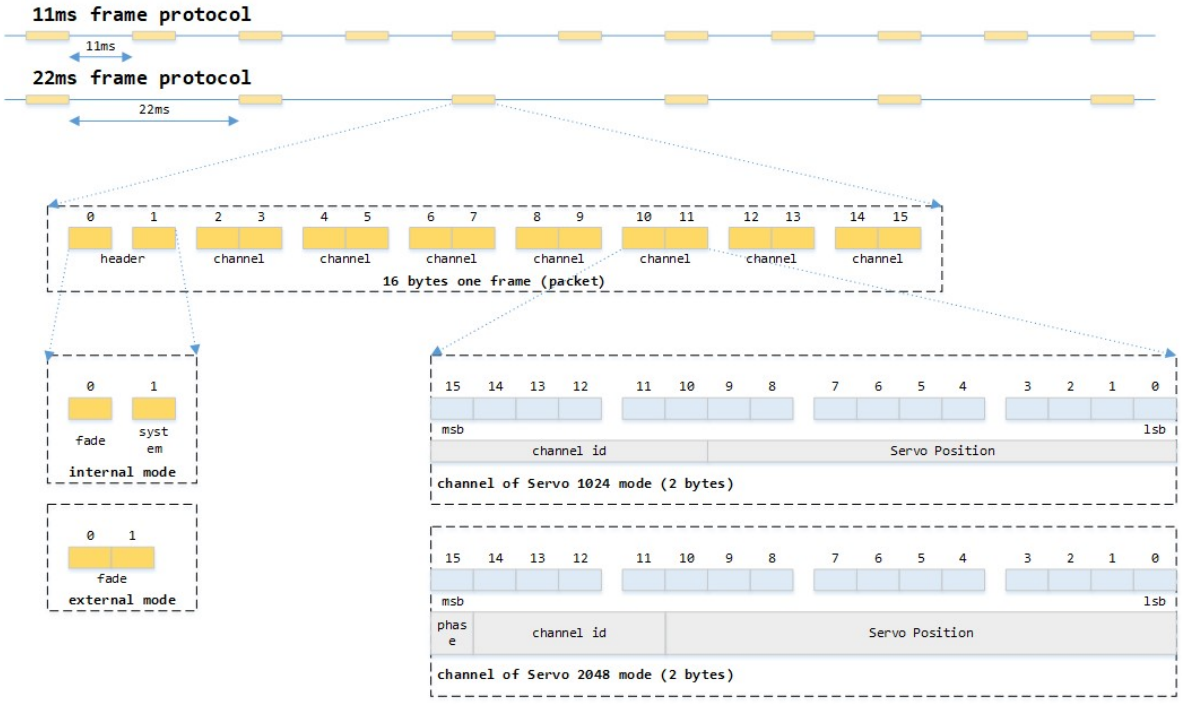
## Data Formats

Specktrum remote receivers는 protocol, 속도, internal mode나 external mode에 따라 다양한 형식으로 데이터를 출력한다.

Internal mode에서의 receiver는 첫 번째 byte(System field)에 사용될 protocol 및 data format을 알 수 있는 system type이 들어 간다. 그리고 두번째 byte에는(fades field) 잃어버린 frame의 갯수를 표시한다. 나머지 14 byte에 data packet이 들어간다.

External mode에서는 첫 2 byte (fades field)에 잃어버린 frame의 갯수를 표시하고, 나머지 14 byte에 data packet이 들어간다.

***All data fields are big-endian, that is, the MSB is transmitted before the LSB. Bit 0 is the lsb, bit 15 is the msb.***




### Servo Field 1024 Mode

DSM2/22ms 모드에서만 사용한다.

### Servo Field 2048 Mode

DSM2/22ms 이외의 모든 모드에서 사용한다.

### Master("Internal Mode")에서의 16 bytes Packet format

```
typedef struct
{
   UINT8 fades;
   UINT8 system;
```

```
   UINT16 servo[7];
} INT_REMOTE_STR;
```

**system field**

| value | protocol | | |
|---|---|---|---|
| 0x01 | 22MS | 1024 | DSM2 |
| 0x12 | 11MS | 2048 | DSM2 |
| 0xA2 | 22MS | 2048 | DSMS |
| 0xB2 | 11MS | 2048 | DSMX |

### Auxilary("External Mode")에서의 16 bytes Packet format

```
typedef struct
{
   UINT16 fades;
   UINT16 servo[7];
} EXT_REMOTE_STR;
```

### Channel ID

16 bytes packet의 각 frame에는 중복된 channel ID가 포함되어서는 안된다. 아래 channel id는 spektrum의 transmitter에서의 channel id의 index이다.

| channel_id | name |
|---|---|
| 0 | Throttle |
| 1 | Aileron(Roll) |
| 2 | Elevator(Pitch) |
| 3 | Rudder(Yaw) |
| 4 | Gear |
| 5 | Aux 1 |
| 6 | Aux 2 |
| 7 | Aux 3 |
| 8 | Aux 4 |
| 9 | Aux 5 |
| 10 | Aux 6 |
| 11 | Aux 7 |

---

## Servo Position Vaules

Servo 데이터는 다음과 같은 규칙으로 해석된다.

### "Servo Position" Ranges

Servo data는 bind type에 의해 결정된다. 즉 1024 모드는 0 ~ 1023이고 2048 모드는 0 ~ 2047 까지이다.

Servo position은 PWM signal 903us ~ 2097us (476 Hz ~ 1107 Hz) 범위에서 1194us(837 Hz) 단계로 변환된다.
조정기의 travel 설정이 100%일때 servo data의 범위는 약 341 ~ 1707이고, 이를 PWM signal로는 1102us(907Hz) ~ 1898us(837Hz) 이다.

**1024 Mode Scaling**

1024 mode에서의 PWM pulse는 다음과 같이 계산된다.

```
PWM_OUT = (ServoPosition x 116.6us) + Offset
```

- PWM_OUT : Pluse length
- ServoPosition : 0 ~ 1023 범위의 값
- 116.6 : 1194 us의 full range에 대한 상수 값 (1194us / 1023 = 1.166 us)
- Offset : 약 1500 us( (903 + 2097) / 2 = 1500 us)

**2048 Mode Scaling**

2048 mode에서의 PWM pulse는 다음과 같이 계산된다.

```
PWM_OUT = (ServoPosition x 58.3us) + Offset
```

- PWM_OUT : Pluse length
- ServoPosition : 0 ~ 2048 범위의 값
- 58.3 : 1194 us의 full range에 대한 상수 값 (1194us / 2048 = 0.583 us)
- Offset : 약 1500 us( (903 + 2097) / 2 = 1500 us)

---

## Cleanflight for STM32F3Discovery

### Configuration

Cleanflight의 configuration에서 UART3 port를 "Serial RX 115200 bps"로 설정하고 Receiver 항목에서 Serial Receiver type을 "spektrum 1024"나 "spektrum2048"로 설정을 한다.

### Bind pin for SPEKTRUM

cleanflight의 STM32F3Discovery board의 target.h 파일에 다음과 같이 PA3를 spektrum bind pin으로 사용한다.

```
#define SPEKTRUM_BIND_PIN       PA3 // USART2, PA3
```

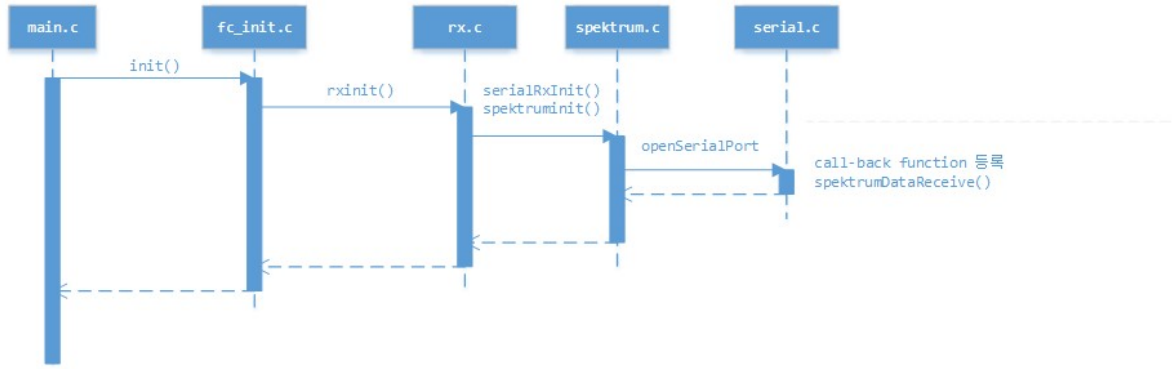
bind pin은 power-on 후에 200 ms 안에 해당 pin의 pulse 갯수로 transmitter에 알려주어야 하기 때문에 부팅 초반에 진행을 해야 한다. 아래는 fc_init.c 파일의 init() 함수에 있는 spektrum bind pin을 검하하기 위한 함수이다.

```
void init(void)
{
   ...
#ifdef SPEKTRUM_BIND_PIN
    if (feature(FEATURE_RX_SERIAL)) {
        switch (rxConfig()->serialrx_provider) {
        case SERIALRX_SPEKTRUM1024:
        case SERIALRX_SPEKTRUM2048:
            // Spektrum satellite binding if enabled on startup.
            // Must be called before that 100ms sleep so that we don't lose satellite's binding window
after startup.
            // The rest of Spektrum initialization will happen later - via spektrumInit()
            spektrumBind(rxConfigMutable());
            break;
        }
    }
#endif
   ...
```

- 위 코드를 분석해 보면 첫번째 SPEKTRUM_BIND_PIN이 정의되어 있어야 build가 되고, bind pin의 검사는 Cleanflight의 configuration을 통해 serial rx가 설정된 port가 존재해야 하고 serial rx에서 SPEKTRUM1024나 SPEKTRUM2048이 설정되어 있어야 한다.

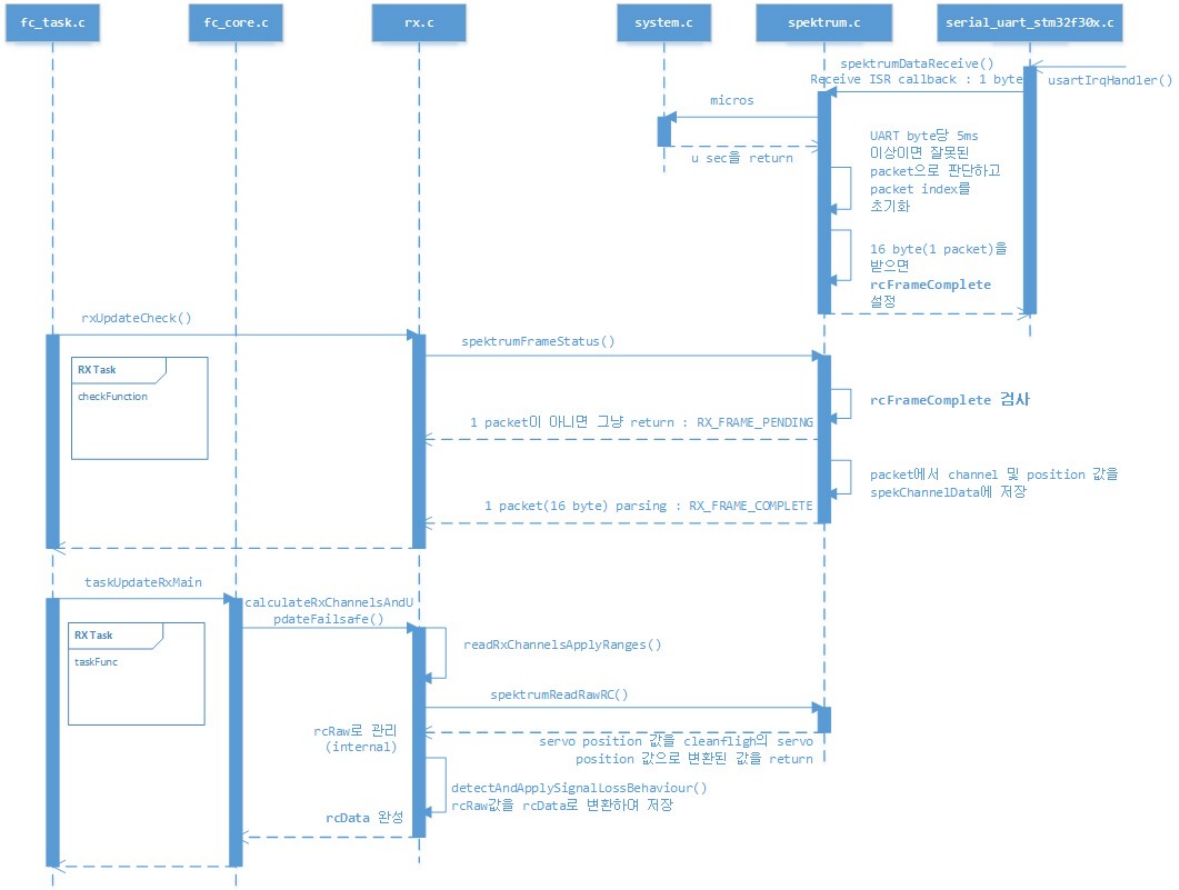
## RX Initialize

Spektrum Serial RX는 configuration에서 설정된 Serial RX UART port로 초기화를 한다. UART를 open할 때 아래의 그림과 같이 "spektrumDataReceive()" call-back 함수를 등록하고 UART를 통해 데이터가 들어오면 해당 call-back 함수가 호출된다.




## Receive RX Data from Receiver

Spektrum1024와 Spektrum2048 mode 동일하게 한 packet size는 16 byte에 최대 7개의 채널을 넣을 수 있다.



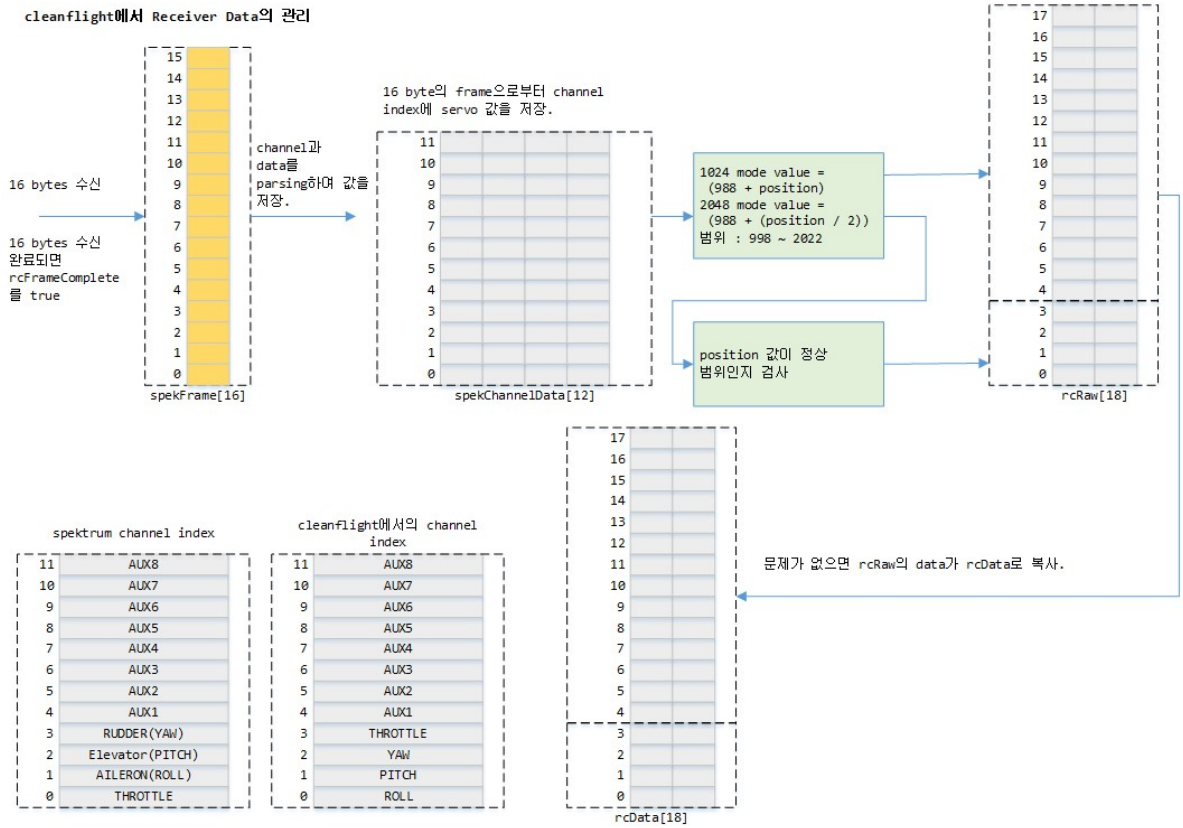

Cleanflight에서는 어떤 이유에서인지 2048 mode에서 11 bits의 resolution을 10 bits로 줄여서 사용한다. 다음은 spektrum.c 파일에 있는 spektrumReadRawRC() 함수이다.

```
static uint16_t spektrumReadRawRC(const rxRuntimeConfig_t *rxRuntimeConfig, uint8_t chan)
{
    uint16_t data;

    if (chan >= rxRuntimeConfig->channelCount) {
        return 0;
    }

    if (spekHiRes)
        data = 988 + (spekChannelData[chan] >> 1);    // 2048 mode
    else
        data = 988 + spekChannelData[chan];           // 1024 mode

    return data;
}
```




- 현재 git에 올라간 source로 동작될 때 channel index가 서로 달라 다르게 동작된다.

---

# Test

test환경은 STM32F3Discovery Board와 Waveshare의 nRF51822 Board를 통해 확인 할 예정이다. 그리고 nRF51822 board에는 nrf51_uart_peripheral firmware에 Spektrum 1024 mode Receiver를 구현한다.

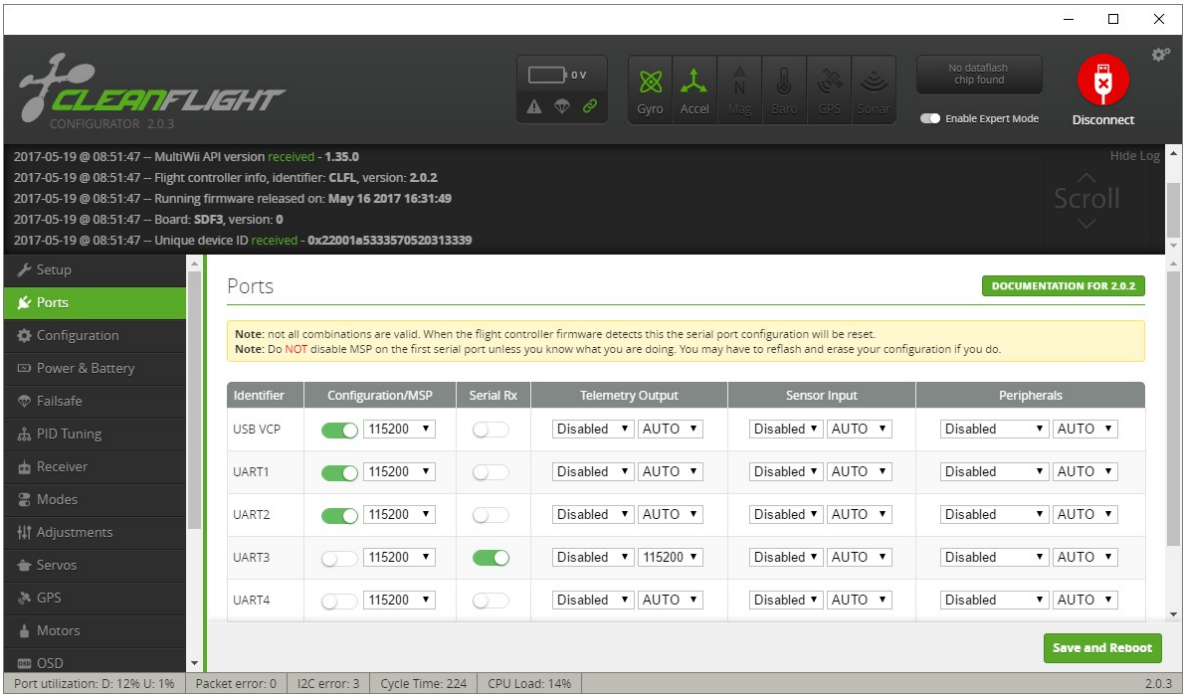
1. **Hardware 연결**

   STM32F3Discover board와 nRF51822 board는 다음과 같이 UART port를 연결한다.

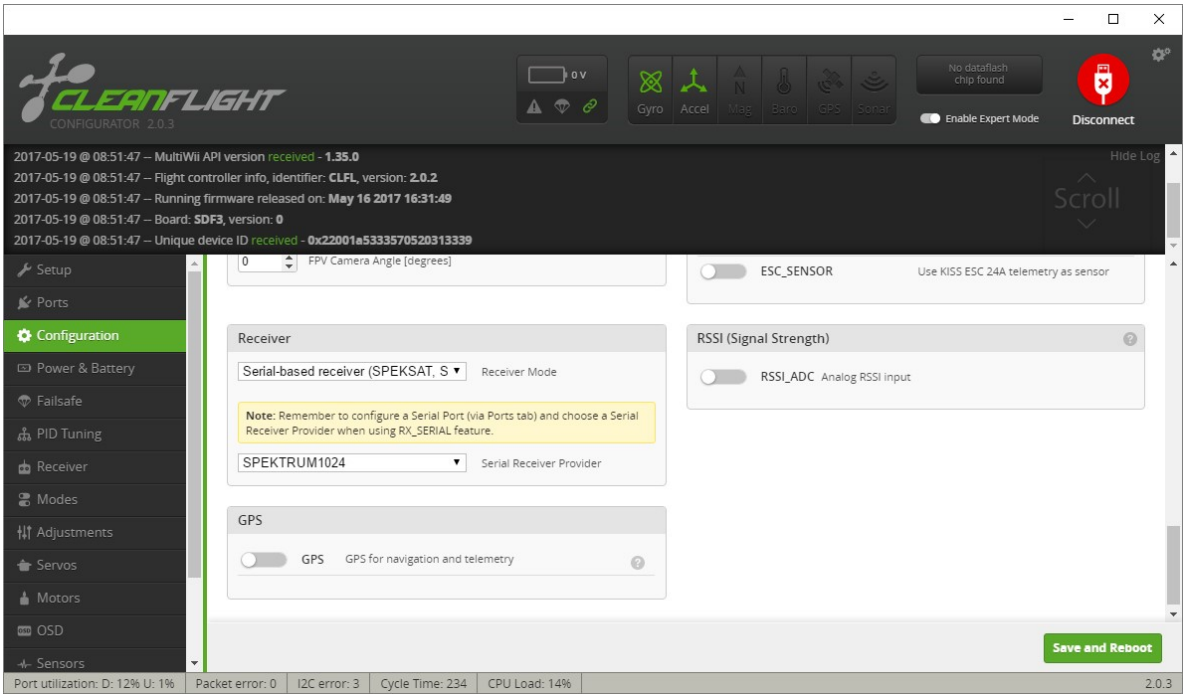
| nRF51822 Board | STM32F3Discovery |
|---|---|
| GND | GND |
| P0.09(TXD) | PB11 (UART3 RX) |
| P0.11(RXD) | PB10 (UART3 TX) - not connect |
| VDD | 3.3V |

2. **Configuration for STM32F3Discovery Board**

   - Ports 항목에서 UART3에서 MSP를 disable(v2.0.3 부터는 MSP와 Serial RX를 동시에 설정하지 못하도록 되었었음.), Serial RX enable 115200 bps, Telemetry disable한다.




- Configuration 항목의 Receiver에서 아래와 같이 "Serial-based receiver (...)"으로 설정하고 Serial Receiver Provider에 "SPEKTRUM1024"를 선택한 후에 "Save and Reboot"을 진행한다.




### 3. nRF51822 BT Receiver(SPEKTRUM1024 mode)

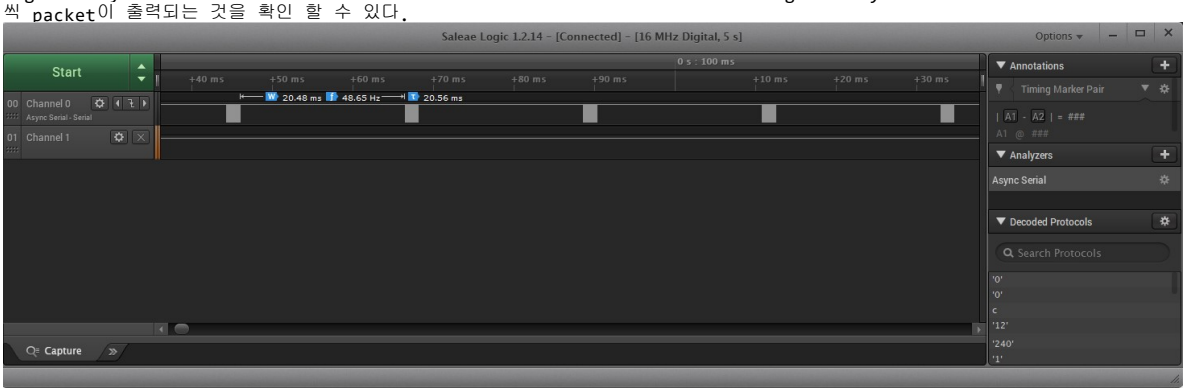
device name은 "RX-SPEKTRUM-1024"

```
static SERIAL_BT_COMMAND g_bt_command[]                 =
{
   { RX_CHANNEL_ROLL,                                   1,      "ROLL-UP"},
   { RX_CHANNEL_ROLL,      SERIAL_BT_COMMAND_DECREASE_FLAG |  1,      "ROLL-DOWN"},
   { RX_CHANNEL_PITCH,                                  1,      "PITCH-UP"},
   { RX_CHANNEL_PITCH,     SERIAL_BT_COMMAND_DECREASE_FLAG |  1,      "PITCH-DOWN"},
   { RX_CHANNEL_YAW,                                    1,      "YAW-UP"},
   { RX_CHANNEL_YAW,       SERIAL_BT_COMMAND_DECREASE_FLAG |  1,      "YAW-DOWN"},
   { RX_CHANNEL_THROTTLE,                               1,      "THROTTLE-UP"},
   { RX_CHANNEL_THROTTLE,  SERIAL_BT_COMMAND_DECREASE_FLAG | 1,      "THROTTLE-DOWN"},
};
```

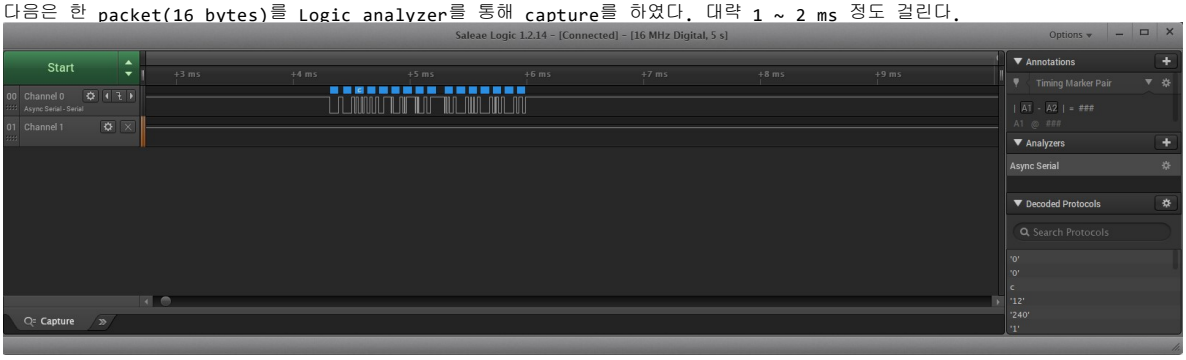
4. **Android nRF Toolbox**

   UART 항목을 선택한 후 버튼에 3번 항에 있는 문자열을 할당한다.
   그리고 BT scan을 한 후 "RX-SPEKTRUM-1024"에 연결을 한다.

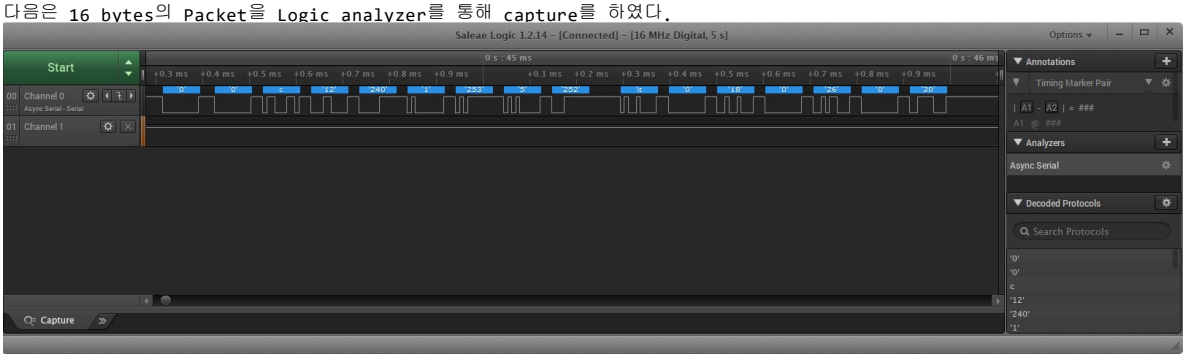
Logic Analyzer를 통해 22ms에 한 번씩 Packet이 출력되는지 확인을 한다. 다음은 logic analyzer를 통해 약 20.48ms에 한 번씩 packet이 출력되는 것을 확인 할 수 있다.




다음은 한 packet(16 bytes)를 Logic analyzer를 통해 capture를 하였다. 대략 1 ~ 2 ms 정도 걸린다.




다음은 16 bytes의 Packet을 Logic analyzer를 통해 capture를 하였다.




specktrum.c 파일에서 보면 프레임 사이는 최소한 5ms 이상이고 packet 안의 byte들 사이는 5ms 이상이 되어서는 안된다. 위 logic analyzer를 통한 capture화면을 보면 모든 조건이 잘 만족되는 것을 확인 할 수 있다.

---

# Debugging

문제점 : nRF51822 board에서 reset이 발생한다.

GDB로 nRF51822 board를 debugging을 하기 위해서는 J-Link가 필요하며, binary를 GDB Debug 용으로 빌드를 다시 해야 한다.

1. Makefile을 수정한다.

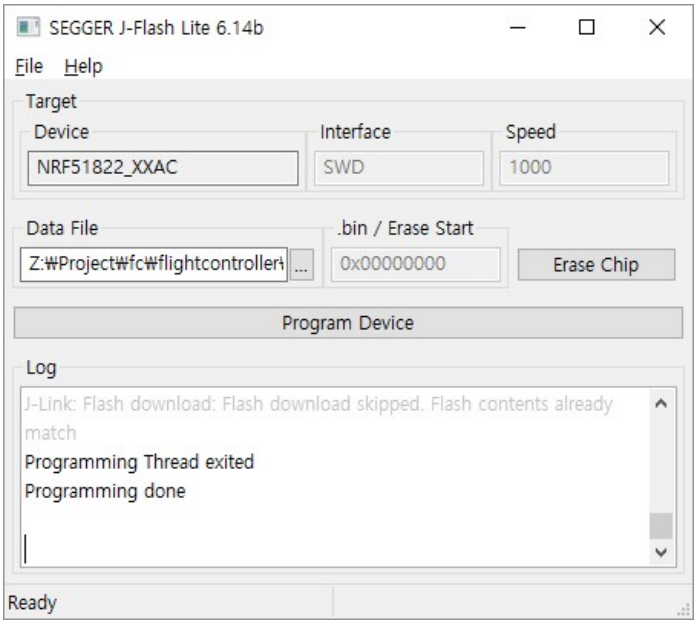
./nRF51_SDK_v11/nrf51_uart_peripheral 에 있는 Makefile의 120라인에 있는 Option을 다음과 같이 수정을 한다.

```
#CFLAGS += -Wall -Werror -O3 -g3^M
CFLAGS += -Wall -Werror -O0 -g^M
```

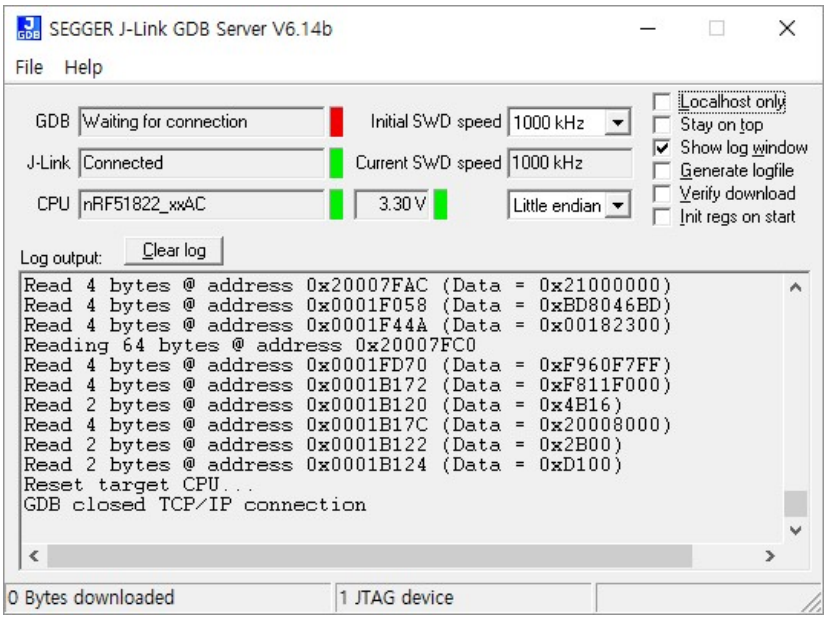
2. peripheral을 빌드한다.

```
./build.sh build=nordic target=s130_peripheral toolchain=6.3.1
```

3. firmware를 J-Link Flash Lite를 통해 download한다.




4. J-link GDB Server를 실행한다.




5. uBuntu command line에서 다음과 같이 gdb를 실행하거나 flightcontroller에 있는 gdb_nordic.sh를 실행한다.

```
cd nRF51_SDK_v11
../toolchain/linux/gcc-arm-none-eabi-6-2017-q1-update/bin/arm-none-eabi-gdb
./nrf51_uart_peripheral/_build/nrf51_uart_peripheral.out
```

GDB를 실행하면 다음과 같은 Message가 display 되고 gdb console이 실행된다.

```
GNU gdb (GNU Tools for ARM Embedded Processors 6-2017-q1-update) 7.12.1.20170215-git
Copyright (C) 2017 Free Software Foundation, Inc.
```

```
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "--host=x86_64-linux-gnu --target=arm-none-eabi".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./nrf51_uart_peripheral/_build/nrf51_uart_peripheral.out...done.
(gdb)
```

위 화면은 gdb가 실행된 것이지 실제 target board는 windows에 연결되어 있기 때문에 remote로 target board에 연결해야 한다. 다음과 같이 연결을 한다.(여기서 ip는 j-link server가 실행 중인 windows의 ip이며 port 번호는 default로 2331이다.)

```
(gdb) target remote 192.168.0.50:2331
Remote debugging using 192.168.0.50:2331
0x00000000 in ?? ()
```

위와 같이 0x00000000 번지에서 stop 되어 있다. 여기서 "continue"를 실행하면 다음과 같이 실행되며 커서는 gdb 프롬프트가 나오지 않고 멈추어져 있다.

```
(gdb) c
Continuing.
```

reset이 발생하면 다음과 같은 Message를 display하며 (gdb) 프롬프트 상태가 된다.

```
(gdb) c
Continuing.

Program received signal SIGTRAP, Trace/breakpoint trap.
0xfffffffe in ?? ()
(gdb)
```

```
$3 = {fades = 0 '\000', system = 0 '\000', channel = {3172, 512, 1536, 2560, 4608, 6656, 7680}}

-> logic analyzer
[000] [000] [100] [012] [000] [002] [000] [006] [000] [010] [000] [018] [000] [026] [000] [030]
 [00]  [00]  [64]  [0C]  [00]  [02]  [00]  [06]  [00]  [0A]  [00]  [12]  [00]  [1A]  [00]  [1E]
-> from gdb
 [0000]      [0C64]      [0200]      [0600]      [0A00]      [1200]      [1A00]      [1E00]


-> logic analyzer
[000] [000] [100] [012] [000] [002] [000] [006] [000] [010] [000] [018] [000] [026] [000] [030]
 [00]  [00]  [64]  [0C]  [00]  [02]  [00]  [06]  [00]  [0A]  [00]  [12]  [00]  [1A]  [00]  [1E]
-> from gdb
 [0000]      [0C64]      [0200]      [0600]      [0A00]      [1200]      [1A00]      [1E00]
```

어떤 함수에서 reset이 발생되었는지 확인하기 위해서는 bt command를 통해 stack을 확인해야 한다.

```
(gdb) bt
#0  0xfffffffe in ?? ()
#1  <signal handler called>
#2  0x000000fe in ?? ()
#3  <signal handler called>
#4  0x0001d686 in nrf_uart_event_clear (p_reg=0x2000232c <m_tx_fifo>, event=0)
    at
/home/duvallee.lee/Project/fc/flightcontroller/nRF51_SDK_v11/components/drivers_nrf/hal/nrf_uart.h:348
#5  0x0001db30 in nrf_drv_uart_tx_for_uart ()
    at
/home/duvallee.lee/Project/fc/flightcontroller/nRF51_SDK_v11/components/drivers_nrf/uart/nrf_drv_uart.c:271
#6  0x0001dc1e in nrf_drv_uart_tx (p_data=0x20002314 <tx_buffer> "", length=1 '\001')
    at
/home/duvallee.lee/Project/fc/flightcontroller/nRF51_SDK_v11/components/drivers_nrf/uart/nrf_drv_uart.c:368
#7  0x0001c866 in uart_event_handler (p_event=0x20007f50, p_context=0x0)
    at
/home/duvallee.lee/Project/fc/flightcontroller/nRF51_SDK_v11/components/libraries/uart/app_uart_fifo.c:76
#8  0x0001df8e in tx_done_event (bytes=1 '\001')
    at
/home/duvallee.lee/Project/fc/flightcontroller/nRF51_SDK_v11/components/drivers_nrf/uart/nrf_drv_uart.c:644
#9  0x0001e138 in uart_irq_handler () at
/home/duvallee.lee/Project/fc/flightcontroller/nRF51_SDK_v11/components/drivers_nrf/uart/nrf_drv_uart.c:752
#10 0x0001e1a4 in UART0_IRQHandler () at
```

```
/home/duvallee.lee/Project/fc/flightcontroller/nRF51_SDK_v11/components/drivers_nrf/uart/nrf_drv_uart.c:847
#11 <signal handler called>
#12 0x0001f058 in __sd_nvic_irq_enable ()
    at
/home/duvallee.lee/Project/fc/flightcontroller/nRF51_SDK_v11/components/softdevice/s130/headers/nrf_nvic.h:156
#13 0x0001f44a in send_receiver_command () at main.c:362
#14 0x0001fd70 in main () at main.c:1289
```

---

# FC - BT Transmitter간 H/W 연결 (v.0.1.0) - 2017-07-07

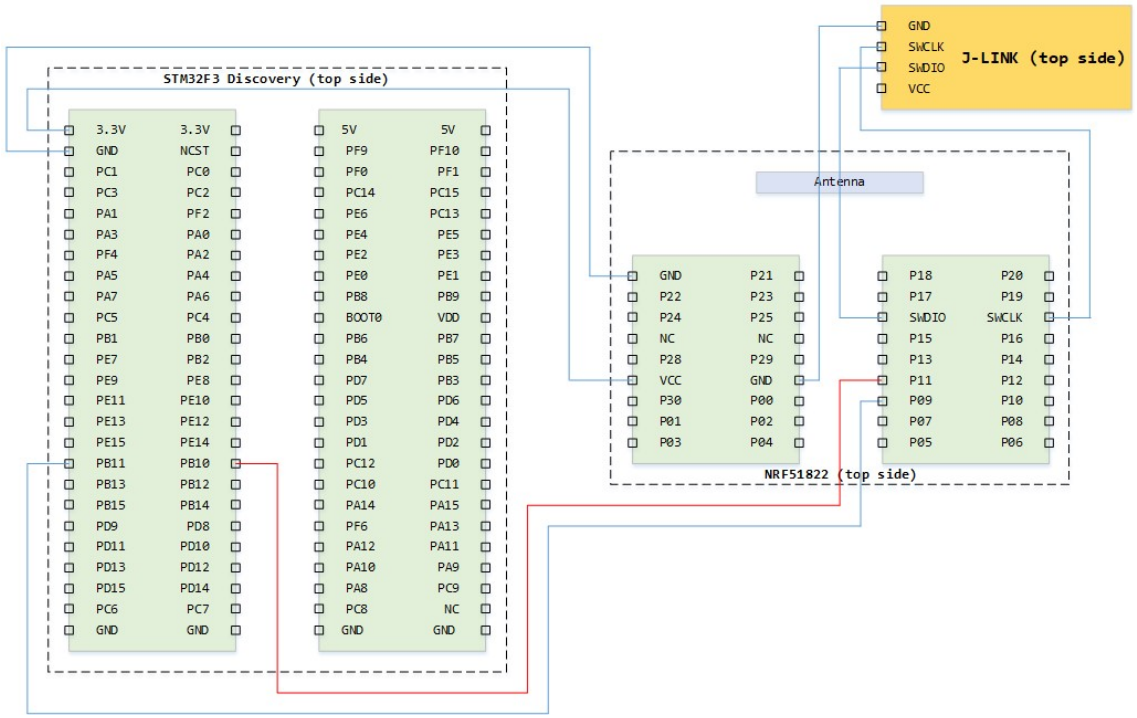
1. 기본 연결

   . 추후 테스트를 위해 RXD - TX로 연결한다.

   | nRF51822 Board | STM32F3Discovery |
   |---|---|
   | GND | GND |
   | P0.09(TXD) | PB11 (UART3 RX) |
   | P0.11(RXD) | PB10 (UART3 TX) - not connect |
   | VDD | 3.3V |

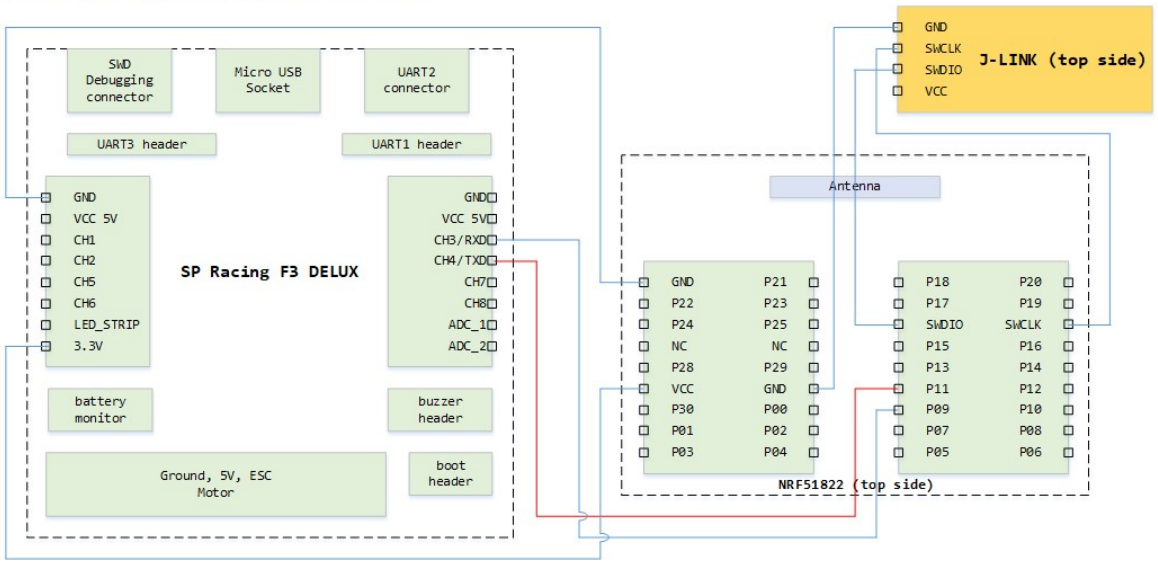
2. **STM32F3Discovery Board**

**STM32F3 Discovery and Nordic NRF51822**




3. **SPRACINGF3 DELUX Board**

**SP Racing F3 Delux and Nordic NRF51822**




---

# Android Phone과 BT Transmitter 간 Protocol 정의(v.0.1.0) - 2017-07-07

## Protocol 시나리오

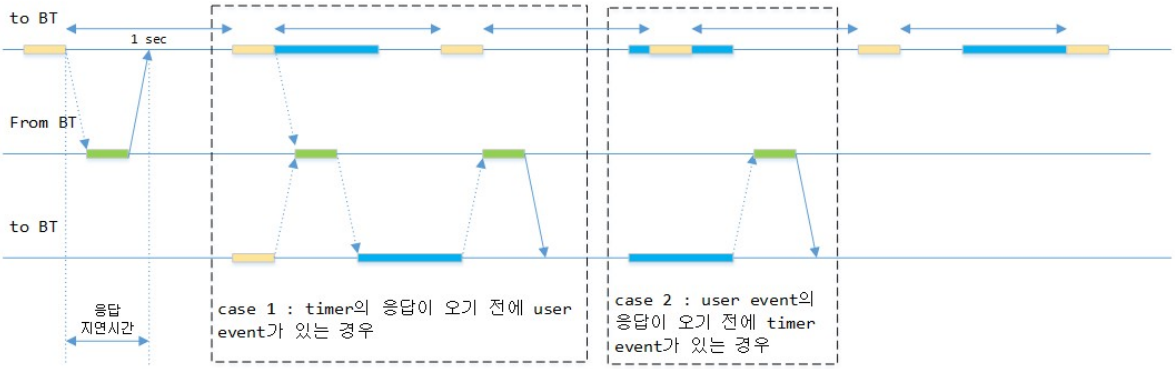
Android Phone에서 BT(nRF51822) 모듈이 alive 되어 있는지 확인을 하기 위해 timer(가칭 1초)를 통해 alive 메세지를 BT Transmitter에 보내고 응답(response)를 통해 alive를 검사한다. 그리고 Android의 UI를 통해 사용자 액션이 발생할때 마다 BT Transmitter에 이벤트를 보내고 응답을 받는다.

**고려해야 할 사항.**

1. 타이머에 의한 alive 메세지를 보낸 후에 BT Transmitter로 부터 응답(response)를 받기 전에 사용자의 액션이 발생한 경우의 처리
2. 사용자의 액션에 의한 메세지를 보낸 후에 BT Transmitter로 부터 응답(response)를 받기 전에 타이머 이벤트가 발생한 경우.
3. command를 보낸 후 응답시간을 얼마나 기다려야 하는가? 를 결정해야 한다.

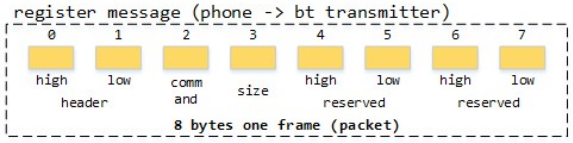
- 위 고려 사항은 Android가 User Event 처리와 타이머가 멀티프로세싱이 된다는 가정이다.
- 위 조건이 맞을 때 Android에서의 Mutex 처리에 관한 검토가 필요하다.
- Android의 타이머 콜백(call-back) 함수를 사용자가 잡고 있을 때 타이머가 지연되는지에 대한 검토가 필요하다.
- channel id는 기존에는 BT module에서 변환하였으나 Android Phone과 CleanFlight의 Channel ID를 동기화하여 BT는 bypass를 하도록 한다.

**Phone과 BT Transmitter 간 Protocol**



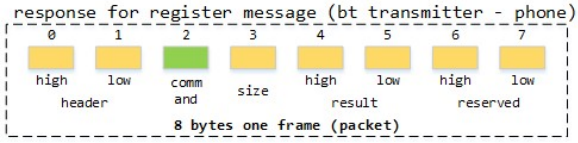
## Phone과 BT Transmitter 간 Protocol v.0.1.0

register message (phone -> bt transmitter)



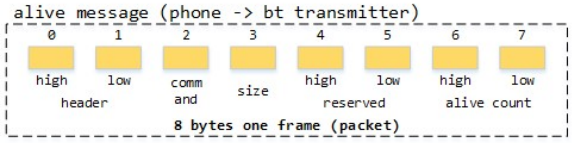

```
header      : high (major & minor version of protocol) - 0x01
              low (sub version of protocol)            - 0x00
command     : 0x01 (register command)
alive count : phone에서 alive message를 보낼때 마다 counting을 한다.
 * version은 protocol version 이다.
 * 연결을 할때마다 맨 처음에 한 번 보낸다.
```

response for register message (bt transmitter - phone)



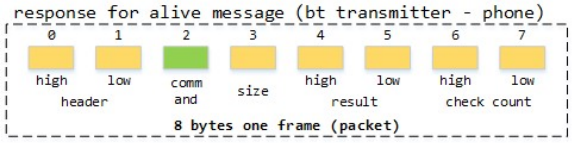

```
header      : high (major & minor version of protocol) - 0x01
              low (sub version of protocol)            - 0x00
command     : 0xF1 (response message for register message)
result      : 0x0 : 성공, 그외의 값 (정의된 error 값)
```

alive message (phone -> bt transmitter)



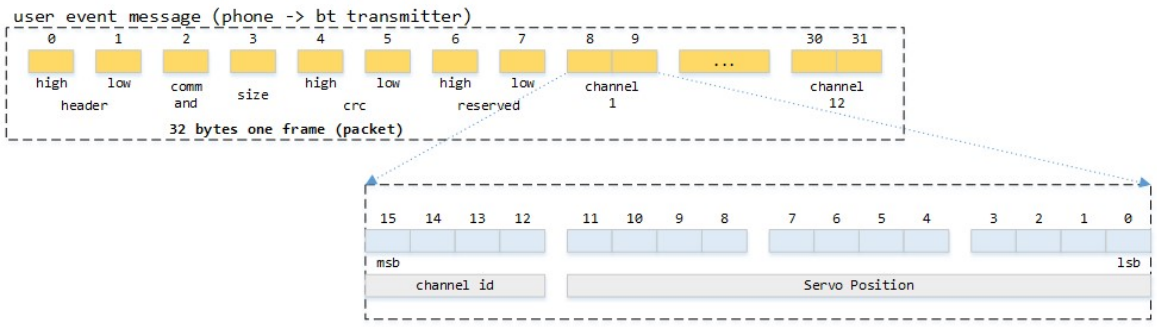

```
header      : high (major & minor version of protocol) - 0x01
              low (sub version of protocol)            - 0x00
command     : 0x02 (alive command)
alive count : phone에서 alive message를 보낼때 마다 counting을 한다.
 * 1초에 한 번씩 보낸다.
```

response for alive message (bt transmitter - phone)



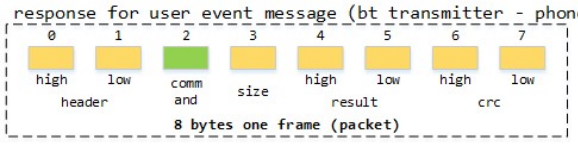

```
header      : high (major & minor version of protocol) - 0x01
              low (sub version of protocol)            - 0x00
command     : 0xF2 (response message for alive message)
result      : 0x0 : 성공, 그외의 값 (정의된 error 값)
check count : register msg를 받은 후에 alive message를 받은 count
```

user event message (phone -> bt transmitter)




```
header      : high (major & minor version of protocol) - 0x01
              low (sub version of protocol)            - 0x00
command     : 0x03 (channel command)
```

response for user event message (bt transmitter - phone)




```
header      : high (major & minor version of protocol) - 0x01
              low (sub version of protocol)            - 0x00
command     : 0xF3 (response message for user event)
result      : 0x0 : 성공, 그외의 값 (정의된 error 값)
crc         : user event의 crc 값
```