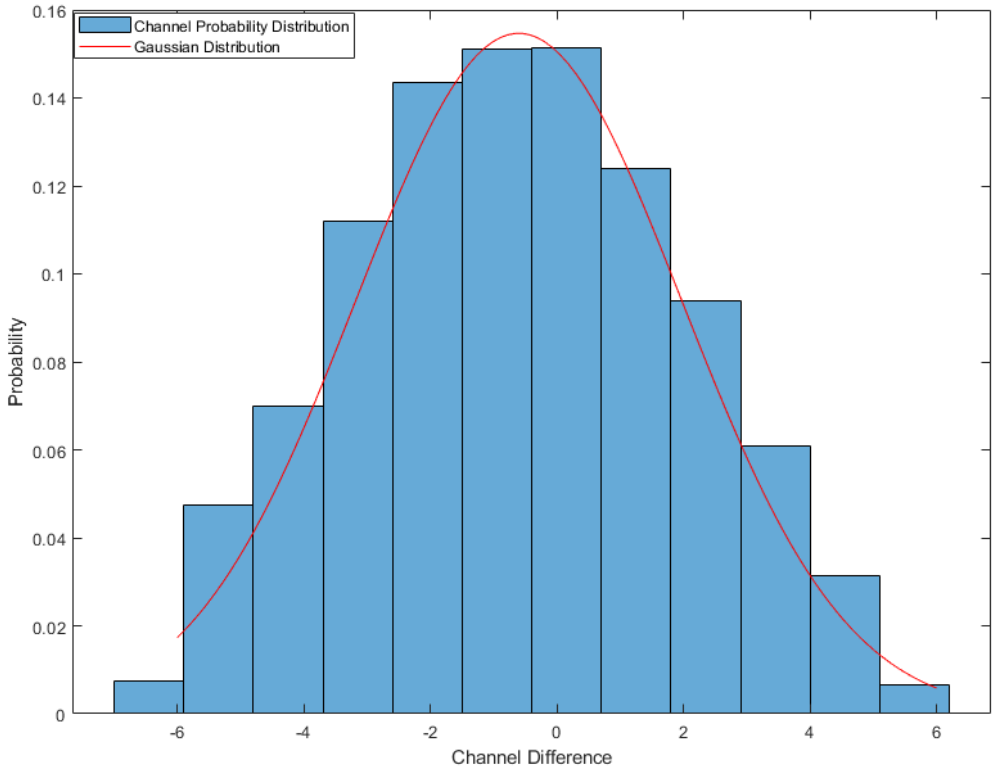
BPS
20210730 seminar

Calibration for channel reciprocity & Documentation of code

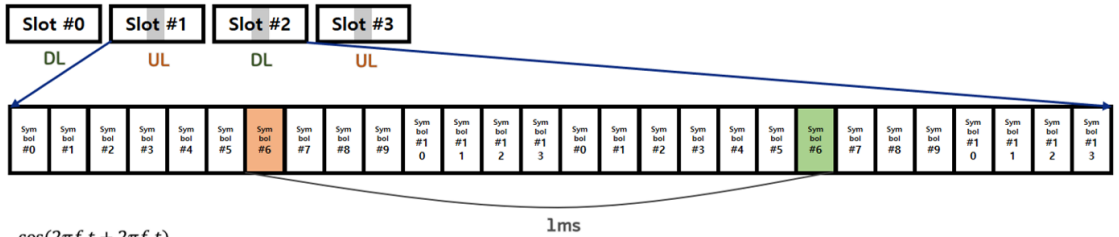
발표자 : 김상은, 유제인

CONTENTS

1.1 Interim conclusion



원인 1) 1ms 동안 phase가 많이 돌아가기 때문에 frequency offset이 발생하여 각도의 차이가 발생함



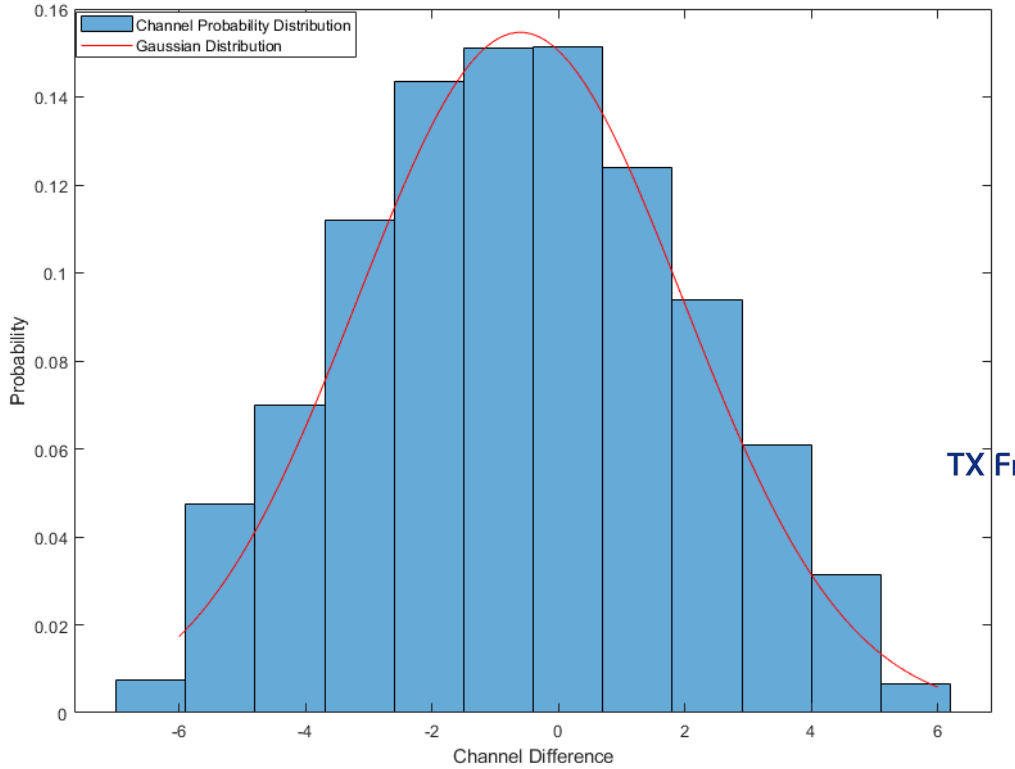
$\cos(2\pi f_c t + 2\pi f_e t)$

$\Delta\theta = 2\pi f_e t$ 이 때, $t = 1ms$, $f_e = -300 \sim 300Hz$ 면, $\Delta\theta = -1.89 \sim 1.89\ rad$

원인 2) RX에서는 frequency offset을 보정해주는 코드가 있는 반면, TX에서는 frequency offset을 보정해주는 코드가 없어서 이러한 오차 값이 발생함

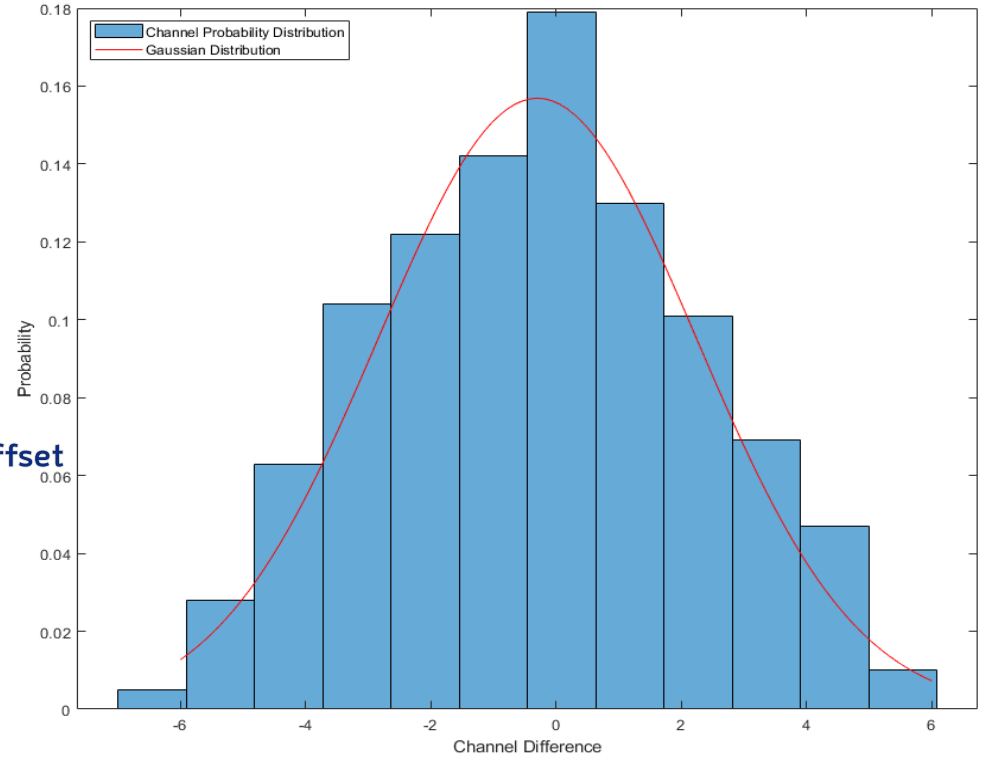
1.1 Interim conclusion

결과 : Channel Non-reciprocity



평균 : -0.6044
표준편차 : 2.5788
표본 : 2000개

TX Frequency offset
보정 후



평균 : -0.2993
표준편차 : 2.5428
표본 : 1000개

1.2 non-reciprocity 원인

결과 : Channel Non-reciprocity

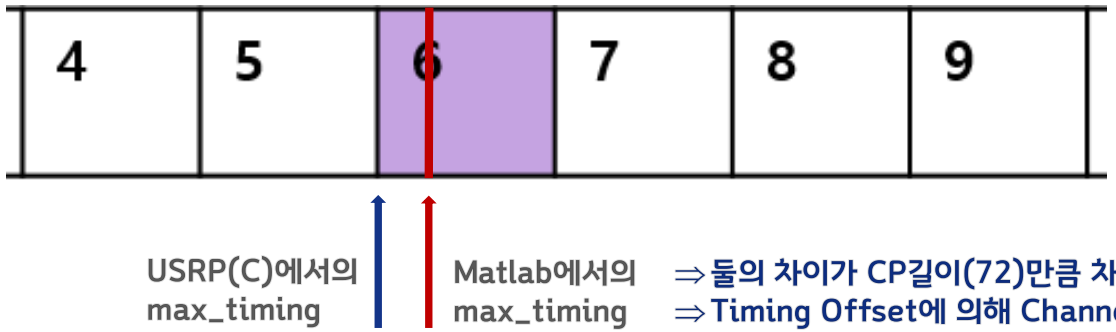
(관련 논문)
Senay Haile, "Investigation of Channel Reciprocity for OFDM TDD Systems", Waterloo, Ontario, Canada, 2009

논문에서는 Channel Non-reciprocity의 원인을 총 4가지로 설명함.

1. Carrier Frequency Offset
2. Timing Offset
3. Sampling Clock Deviations
4. Other Detriments

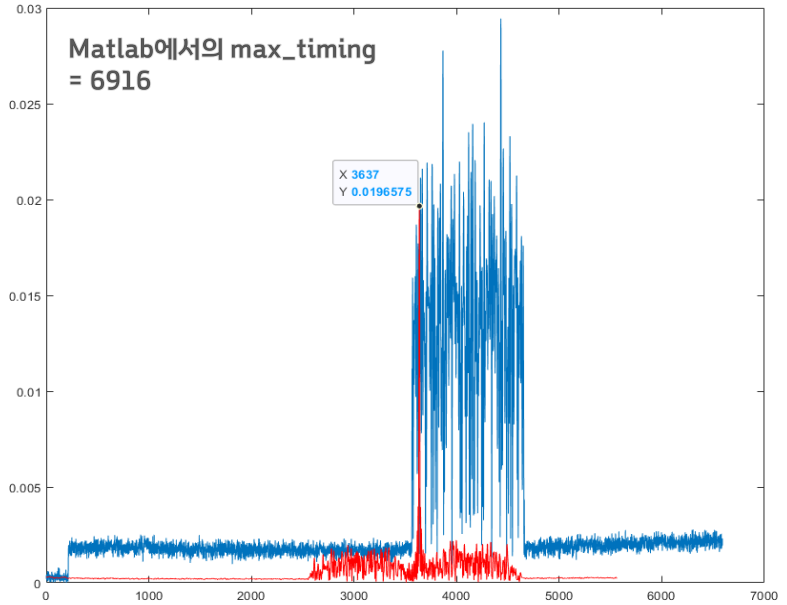
Slot #1&3

0	1	2	3	4	5	6	7	8	9	10	11	12	13



USRP(C)에서의 max_timing = 6844

```
[PHY]  [VRX] WARNING ::: too large frequency off
sated immediately
[PHY] detection success (time offset : 6844)
slot_nr : 2 ,count1024: 907
[PHY]  [VRX] WARNING ::: too large frequency off
sated immediately
[PHY] detection success (time offset : 6844)
slot_nr : 0 ,count1024: 908
[PHY]  [VRX] WARNING ::: too large frequency off
sated immediately
[PHY] detection success (time offset : 6844)
slot_nr : 2 ,count1024: 909
[PHY]  [VRX] WARNING ::: too large frequency off
sated immediately
[PHY] detection success (time offset : 6844)
slot_nr : 0 ,count1024: 910
```



⇒둘의 차이가 CP길이(72)만큼 차이가 발생함.
⇒Timing Offset에 의해 Channel Non-reciprocity가 발생한다고 생각함.

2.1 markdown

3.1 NPHY_nr-ue

vThread_L1 Protocol 설명

Symbol or Bit Data Send Method

SER or BER Performance calculation

: Symbol or Bit Data를 보내기 위해 필요한 함수 및 변수 설명

: NPHY_thread_L1에서 Bit Data를 처리하는 과정 및 코드 설명

코드 구현 (C)

1. Symbol or Bit Data를 보내기 위해 필요한 함수 및 변수 설명

1-1) 변수 설명

```
#define bitsend
#define symbolsend
#define DBGBER
```

→ Bit Data를 보내고 싶은 경우, bitsend 변수 켜주기

→ Symbol Data를 보내고 싶은 경우, symbolsend 변수 켜주기

→ BER를 디버깅하고 싶은 경우, DBGBER 변수 켜주기

(BER은 STLC에서만 확인해 봄. SISO, STBC의 경우에 이상하면 변수켜서 확인하기)

```
//symbol data buffer
short txdatabuf[100]={23169, -23169, 23169, 23169, -23169, -23169, -23169, 23169, 23169, 23169,...}
int dataLength = 10;

//bit data buffer
short txbitbuf[NPHY_FFTSIZE][NPHY_BITDATASize] ={0,};
short txbitbuf10[NPHY_BITDATASize]={0,};
```

→ txdatabuf는 Symbol Data, txbitbuf는 Bit Data 정보를 저장하고 있는 Buffer

→ txdatabuf는 data를 채워둔 후, dataLength라는 변수를 이용하여 길이를 조절할 수 있음

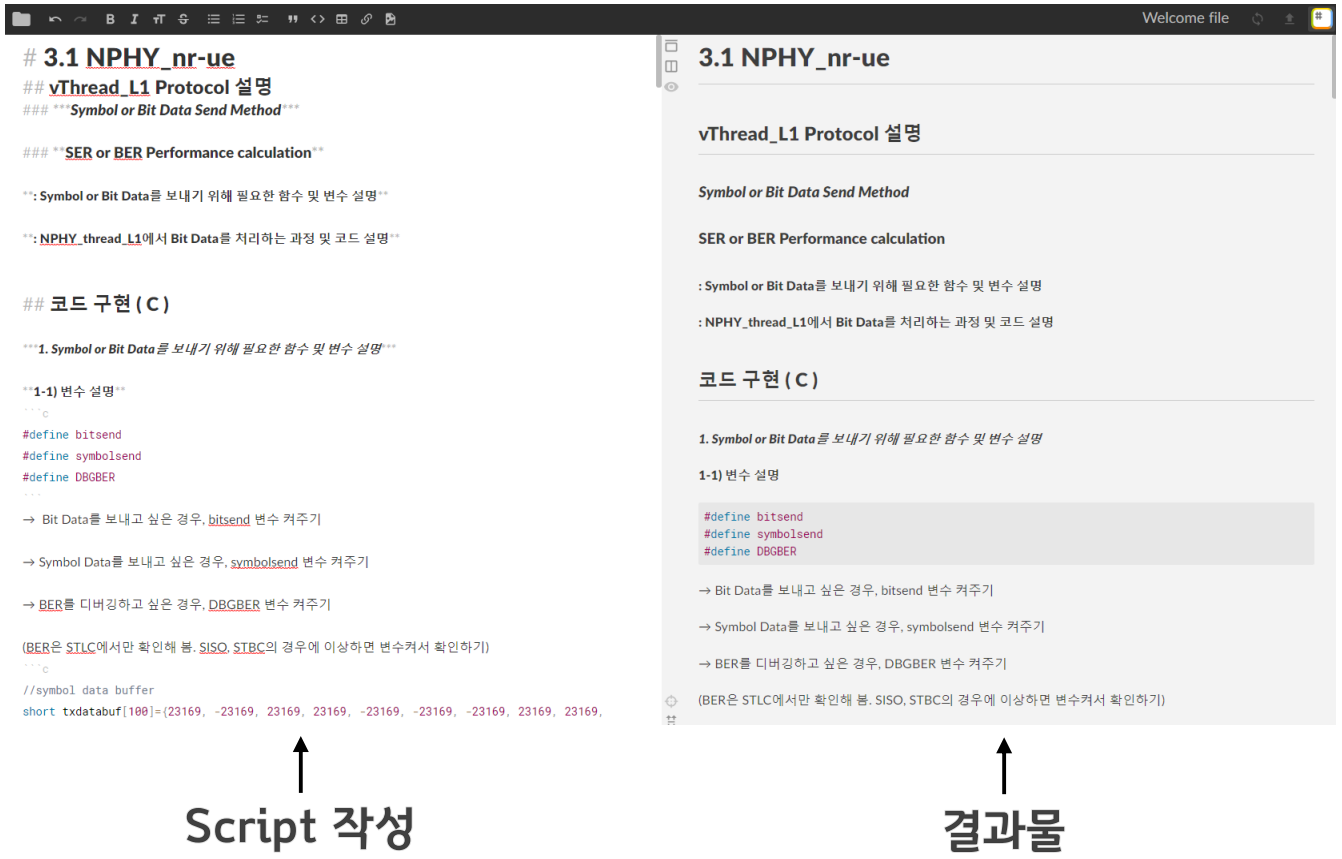
→ NPHY_FFTSIZE = 1024으로 data의 총 개수인 2의 10승, 즉 1024가지의 데이터를 담을 수 있게 설정해 둠

→ NPHY_BITDATASize = 10으로 data 1개를 나타내는 bit가 10bit임을 나타내 줌

1-2) void gen_bitdata()

> Data를 10 bit로 0 ~ 1023을 표현하여 txbitbuf에 넣어줌
>
> ex) 1023 = 1 1 1 1 1 1 1 1 1 1 로 표현가능
>
> p0~p9는 slotCnt를 2진수로 나타냈을 때, 각 자리당 해당하는 숫자를 의미함
>
> (이때, 0부터 차례대로 9번까지 해당하는 숫자를 쓰면 원하는 이진수를 표현하도록 설정해 둠)

Markdown Editor : StackEdit – In-browser Markdown editor



Script 작성

결과물

2.1 markdown

1. Header

```
# 제목 1
## 제목 2
### 제목 3
#### 제목 4
##### 제목 5
###### 제목 6
```

제목 1

제목 2

제목 3

제목 4

제목 5

제목 6

2. Code Block

```c
int year = 2021;
int month = 7;
int day = 30;

printf("Today is %d.%d.%d
\n",year,month,day);
```

```c
int year = 2021;
int month = 7;
int day = 30;

printf("Today is %d.%d.%d \n",year,month,day);
```

```matlab
figure(1);
set(1,'color','w')
histogram(hist_diff,10);
```

```matlab
figure(1);
set(1,'color','w')
histogram(hist_diff,10);
```

프로그래밍 언어 선택 가능

3. 인용 상자

```
>1
>>2
>>>3
```

> 1
>
> > 2
> >
> > > 3

2.1 markdown

4. 강조

기울임꼴

```
1. *BPS*
2. _BPS_
```

1. *BPS*
2. *BPS*

굵게

```
1. **BPS**
2. __BPS__
```

1. **BPS**
2. **BPS**

취소선

```
~~BPS~~
```

~~BPS~~

5. 이미지

```
![](https://www.sookmyung.ac.kr/sites/sookmyungkr/images/sub/contents/ui_symbol_01.png)
```

사이즈 조절

```
<img  src="https://www.sookmyung.ac.kr/sites/sookmyungkr/images/sub/contents/ui_symbol_01.png" width="20%">
```

Html 문법
-> html 형식에서도
변경된 Size 적용 O

※ HTML로 export할 때, 사진이 첨부되기 위해서는?raw=true 추가

ex)

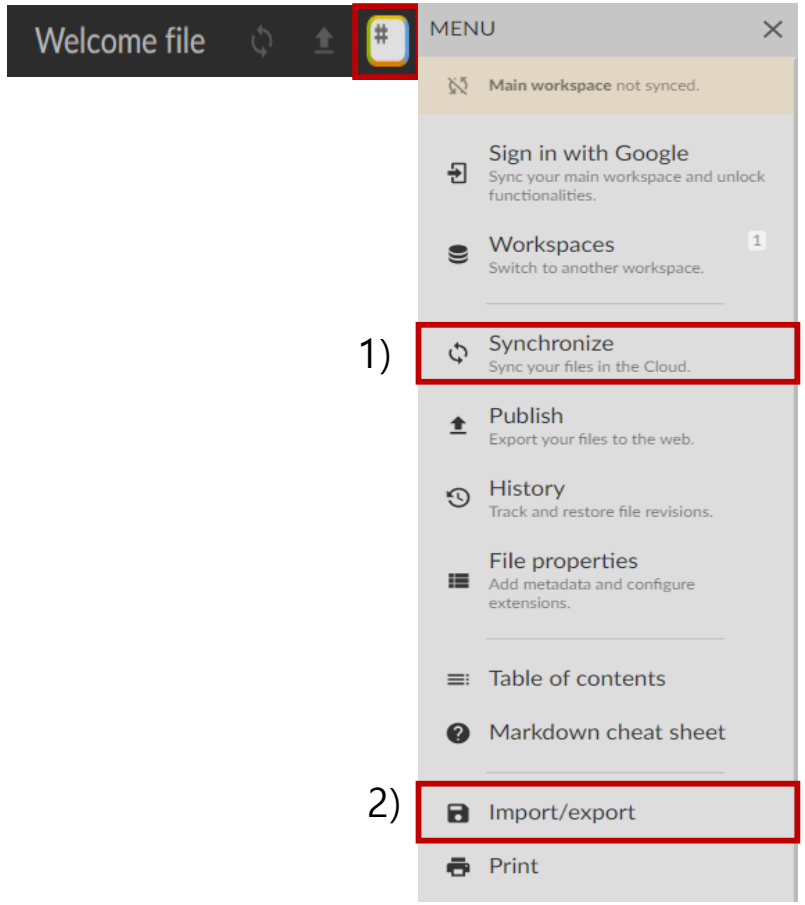
+) 참고

https://guides.github.com/features/mastering-markdown/

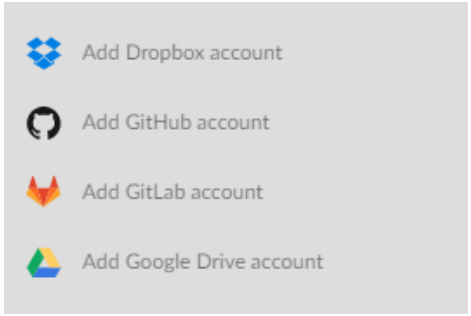
https://heropy.blog/2017/09/30/markdown/

2.1 markdown

파일 저장 방법

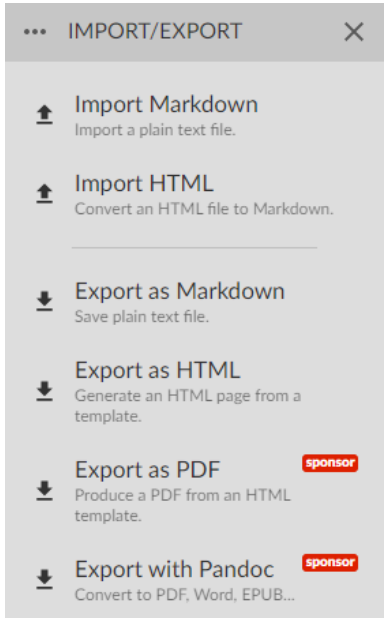


1) 작성된 markdown을 원하는 공간에 저장



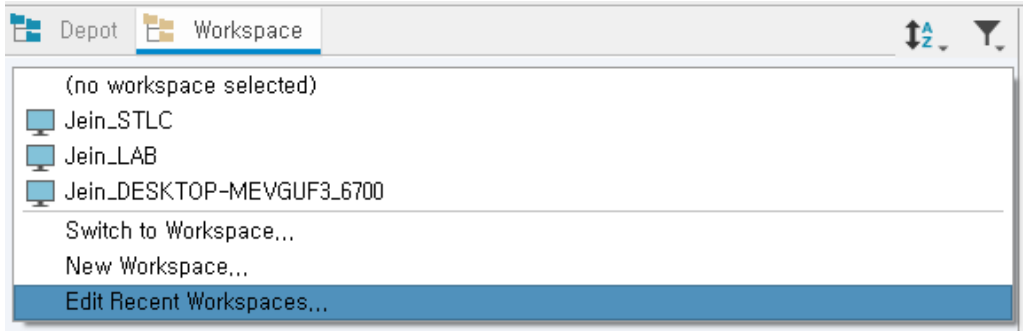
※ 파일 upload가 되지 않을 시
직접 .md 파일로 저장

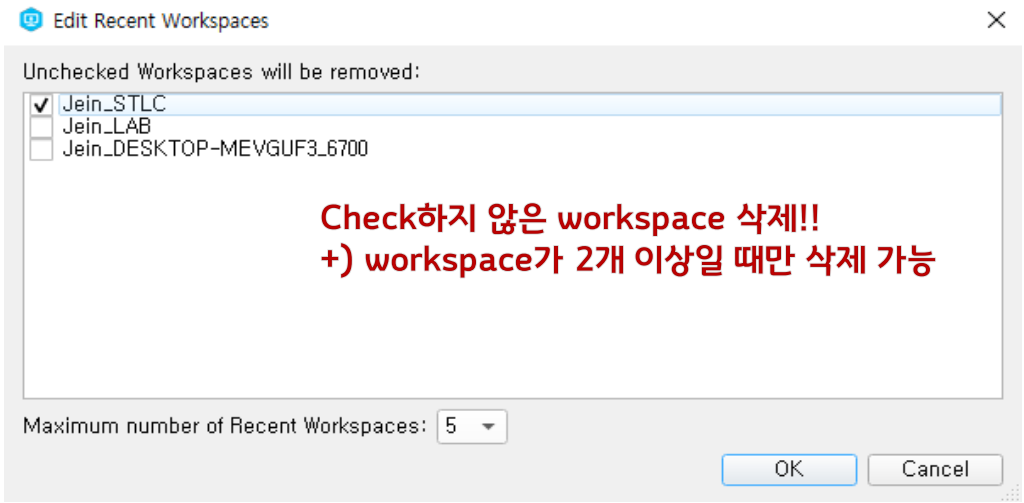
2) HTML or Markdown 형식으로 파일 추출 가능



※ HTML로 export할 때,
사진이 제대로 첨부되었는지 확인 필요
(이미지 로드 방식에 따라 나오지 않을 수 O)

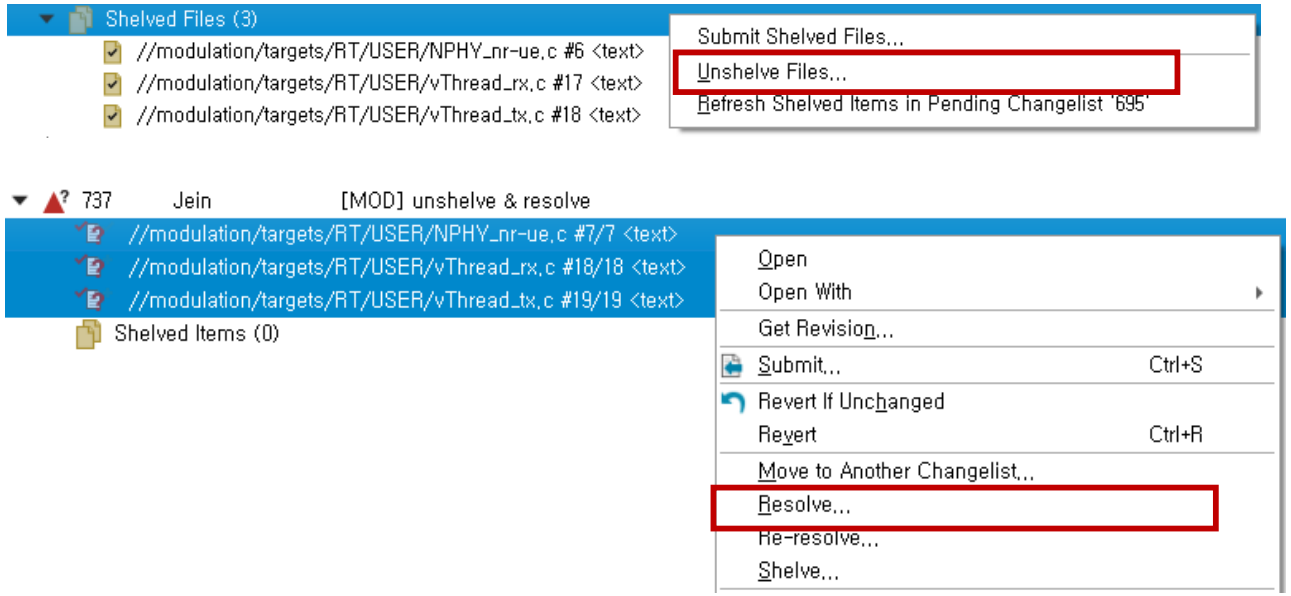
1. Workspace 삭제

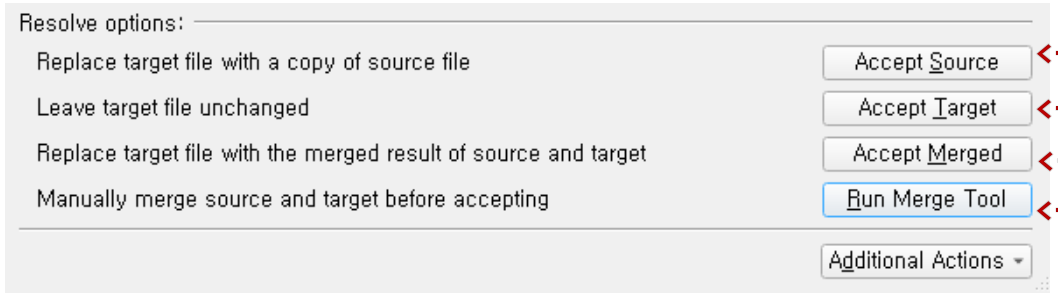




Check하지 않은 workspace 삭제!!
+) workspace가 2개 이상일 때만 삭제 가능

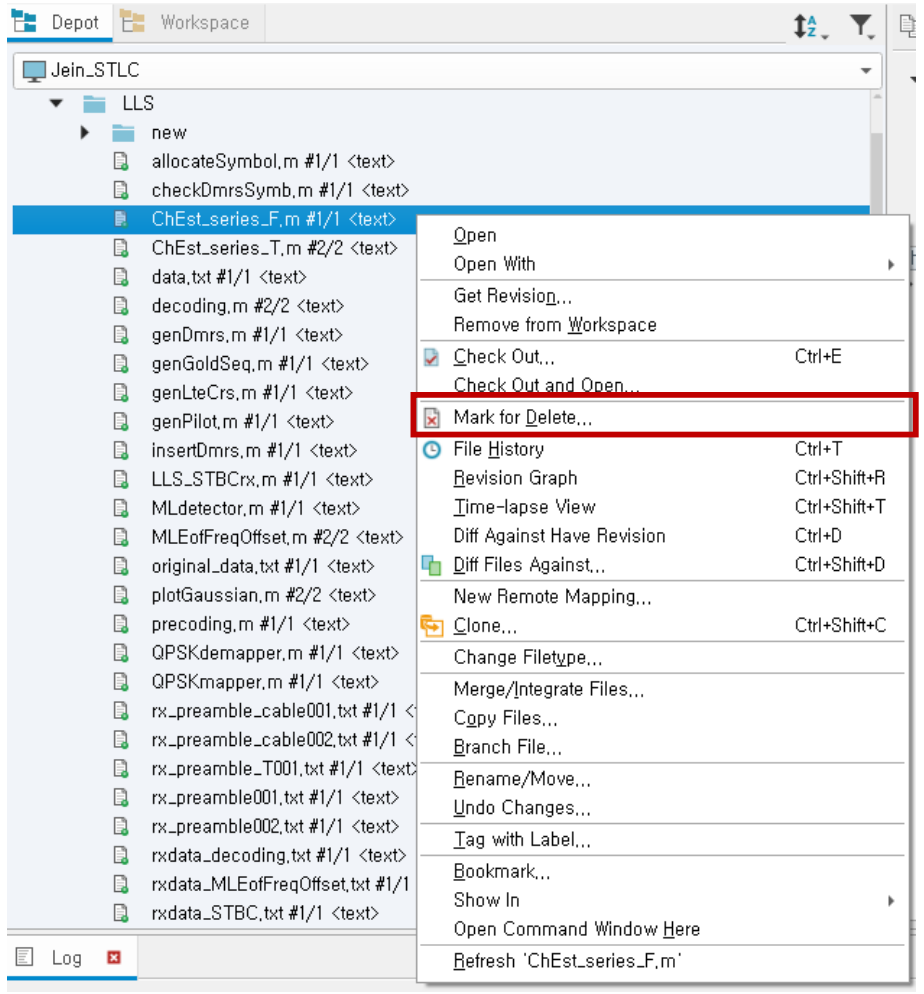
2. Unshelve & Merge file



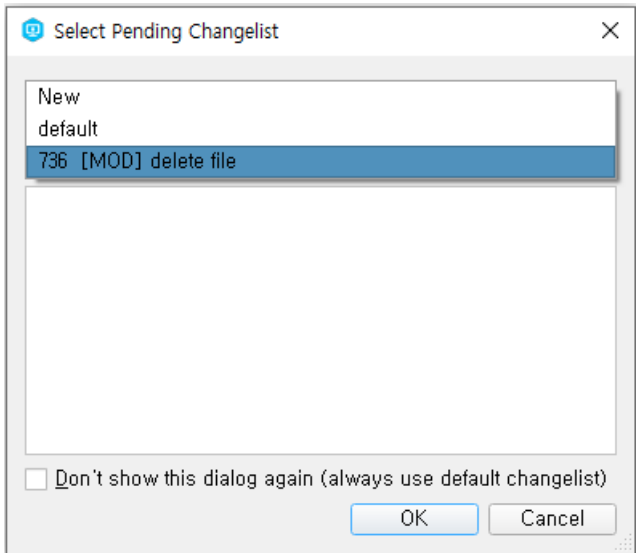


<-덮어쓰기
<-Unshelve 취소
<-Merge
<-Merge 확인

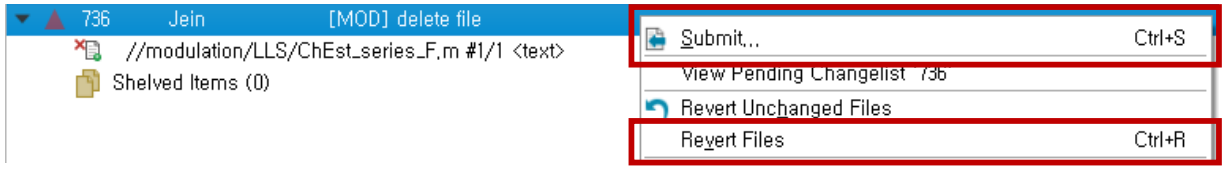
3. Submit한 파일 삭제



파일을 삭제하기 위해 만든 CL 선택



최종 파일 삭제



파일 삭제 취소 -> Revert File

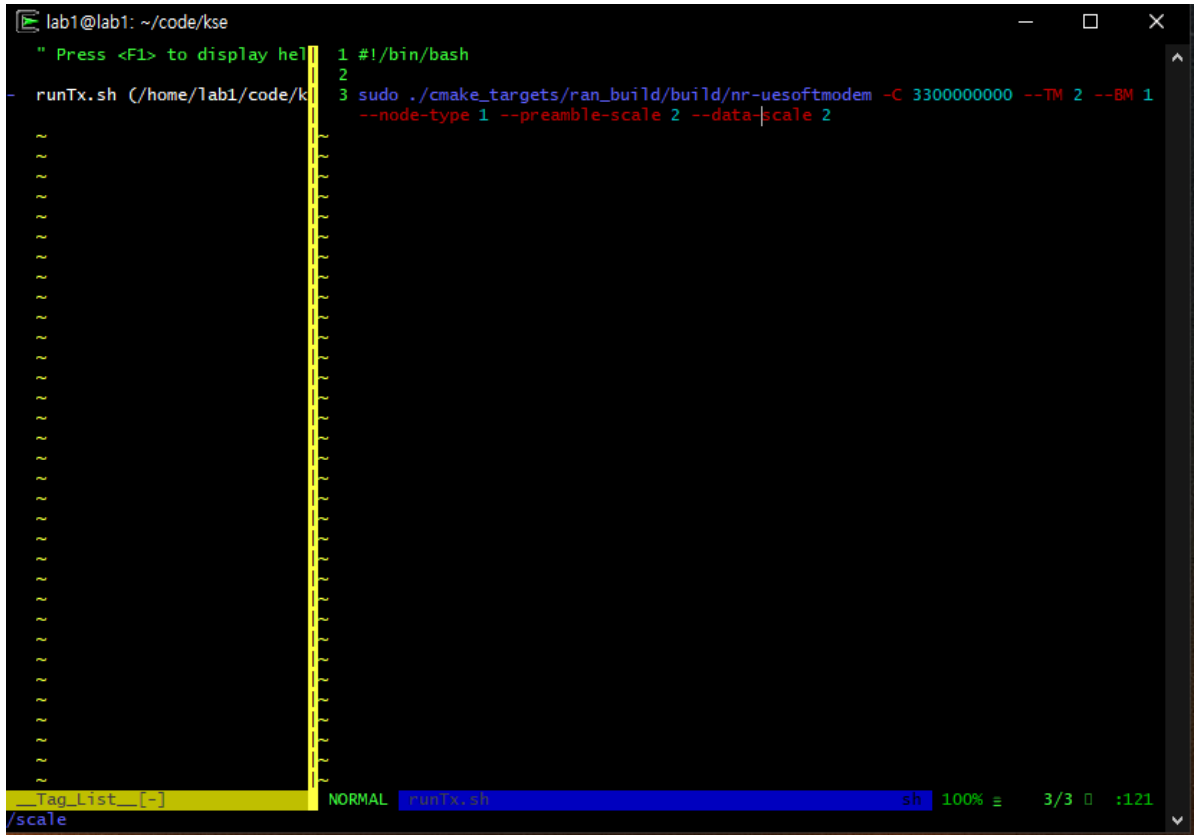
- cd : change director의 약자; 하위 디렉토리로 이동 가능
  (예시) cd code/modulation

- cd .. : 상위 디렉토리로 이동
  (예시) cd ../.. (상위 디렉토리로 두 번 이동)

- cd . : 현재 디렉토리로 이동
  (예시) cd .

- ls: list의 약자; 현재 디렉토리(폴더) 아래의 모든 항목을 보여줌

- vi : visual editor에서 유래; 줄 단위의 편집기가 아닌 한 화면을 편집 가능
  (예시) vi runTX.sh
  ▶ vi 사용법
  → /(검색어) : 파일에서 단어 검색
  (n을 누르면 다음 검색어로 넘어감)
  (N 또는 b를 누르면 이전 검색어로 넘어감)
  (예시) /error
  → :q : 닫기
  → : q! : 변경된 내용이 있더라도 저장하지 않고 무조건 닫기
  → :wq : 쓰고 닫기
  → :wq! : 변경된 내용을 무조건 쓰고 닫기
  → i : 쓰기모드(쓰기 모드 상태에서 esc 누르면 기본 모드로 변경됨)



vi 예시
/(검색어) : 파일에서 단어 검색(n을 누르면 다음 검색어로 넘어감)

1. Data를 추출하는 방법 (Remote_원격지 ⇒Local_컴퓨터)
- 띄어쓰기를 주의해서 써줘야 함

scp 원격지_id@원격지_ip:~/원본위치/파일명 ./

(예시) scp lab1@192.168.1.101:~/code/kse/rxdata.txt ./

(참고) vThread_HWCommon.c 파일에 있는 아래 함수를 이용하여 Dump file을 만들 수 있음
- vhw_dumpDataToFile
- vhw_dumpDataToFileAppend

2. 강제로 읽기 파일을 쓰기 파일로 바꿔주는 방법
- P4V에 Pending해서 수정하는게 좋음. 계속 안 될 때 사용

chmod 775 ./수정할 파일의 위치/파일이름

(예시) chmod 775 ./targets/RT/USER/vThread_HWCommon.c

3. 처음에 파일을 보낼 때(Local_컴퓨터 ⇒ Remote_원격지)
- 리눅스 서버가 컴파일하기 위해서는 전체 파일의 $을 제거하고 보내줘야 함
- Local_컴퓨터에서 아래의 명령어를 입력한 후 원격지로 보내야 함
- Dos2unix 명령어 사용

find ./|xargs dos2unix

4. Memry에 문제가 생겼을 때 확인하는 방법
- 버퍼에 저장된 시간이 뜸. 현재 제대로 저장되었는지 확인 가능

ls -all

```
lab1@lab1:~/code/kse$ ls -all
total 181432
drwxr-xr-x 13 lab1 lab1      4096 Jul 28 05:00 .
drwxrwxr-x 13 lab1 lab1      4096 Jul 27 21:38 ..
-rw-rw-r--  1 lab1 lab1       147 May 18 05:03 @
drwxr-xr-x 12 lab1 lab1      4096 Mar 25 02:04 ?
-rwxrwxr-x  1 lab1 lab1       105 Jun 21 00:58 biSync.sh
-rw-------  1 lab1 lab1     12288 Jun  7 04:27 .biSync.sh.swp
-r-xr-xr-x  1 lab1 lab1       136 Jun  7 04:22 buildtxrx.sh
drwxr-xr-x  4 lab1 lab1      4096 May  5 06:17 ci-scripts
-r-xr-xr-x  1 lab1 lab1        50 Jun  7 04:22 cleanUE.sh
drwxr-xr-x 17 lab1 lab1      4096 May 18 05:29 cmake_targets
drwxr-xr-x  4 lab1 lab1      4096 Jul  8 04:29 common
-rw-------  1 root root 112394240 May 17 04:20 core
-r-xr-xr-x  1 lab1 lab1        65 Jun  7 04:22 gNBsync.sh
-r-xr-xr-x  1 lab1 lab1        72 Jun  7 04:22 InitBuild.sh
-r-xr-xr-x  1 lab1 lab1     12144 Jun  7 04:22 LICENSE
drwxr-xr-x  7 lab1 lab1     45056 Jul 15 09:14 LLS
-r-xr-xr-x  1 lab1 lab1       271 Jun  7 04:22 maketags
drwxr-xr-x  4 lab1 lab1      4096 May  5 06:17 nfapi
-r-xr-xr-x  1 lab1 lab1       902 Jun  7 04:22 NOTICE.txt
-r-xr-xr-x  1 lab1 lab1       762 Jun  7 04:22 oaienv
-r-xr-xr-x  1 lab1 lab1     14622 Jun  7 04:22 OAINR.SearchResults
drwxr-xr-x  9 lab1 lab1      4096 May  5 06:17 openair1
drwxr-xr-x 14 lab1 lab1      4096 Mar 25 01:53 openair2
drwxr-xr-x 12 lab1 lab1      4096 May  5 06:17 openair3
-r-xr-xr-x  1 lab1 lab1       695 Jun  7 04:23 pre-commit
-r-xr-xr-x  1 lab1 lab1      2464 Jun  7 04:23 README.txt
-rwxrwxr-x  1 lab1 lab1       147 Jul 19 03:58 runRx.sh
-rw-r--r--  1 lab1 lab1     12288 Apr 27 04:59 .runRx.sh.swo
-rw-------  1 lab1 lab1     12288 Apr 27 04:54 .runRx.sh.swp
-rwxrwxr-x  1 lab1 lab1       141 Jul 19 03:58 runTx.sh
-rw-r--r--  1 root root     87770 Jul  9 02:54 rxdata0.txt
-rw-r--r--  1 root root    101246 Jul  9 02:54 rxdata1.txt
-rw-r--r--  1 root root    107805 Jul 15 04:41 rxdata.txt
```

언제 만들어진 파일인지 날짜와 시간을 확인할 수 있음

※ Segmentation Fault 문제 디버깅하는 방법

(상황) USRP에서 Compile이 되었는데 돌아가지 않을 때

⇒ LOG가 어디까지 Print 했는지 확인하기

⇒ 기존 core dump 지우기
rm core

⇒ segment fault가 발생하면 자동으로 core file이 생성되도록 설정
(core의 size를 모르므로 일단 unlimited로 설정)
ulimit –c unlimited

⇒ 다시 실행파일 Run

⇒ Segmentation Fault 문제 발생!

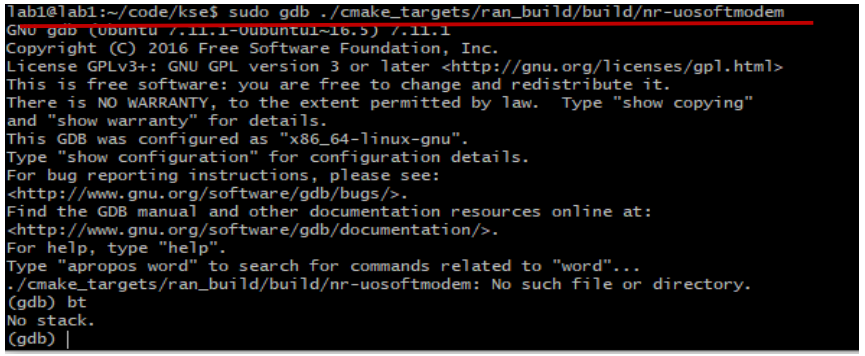
디버깅할 위치
: Segmentation Fault 뒤에 다음과 같은 위치정보가 뜸
: 이를 복사하여 사용해주면 됨

뜨는 문구: (core dumped) sudo ./cmake_targets/ran_build/build/nr-uosoftmodem

입력할 코드: sudo gdb ./cmake_targets/ran_build/build/nr-uosoftmodem –c ./core

⇒ 위의 코드를 순서대로 적어 줌. 그럼 아래와 같이 Frame 번호가 나오며,
어느 위치에서 코드가 멈췄는지 확인할 수 있음.

(예시) #0 vrx_detectPreamble (몇 번째줄)
#1
#2



gdb 화면 (예시 화면은 core가 없는 상태여서 bt를 쳐도 stack이 안 나옴)

▶ Frame 번호가 안뜨고 (gdb)라고 뜰 경우
(gdb) bt

위의 코드를 써주면 #0, #1, #2와 같은 Frame 번호가 뜸

⇒ 이때 가장 최신에 실행된 파일은 #0에 뜸

(gdb) frame frame 번호
(예시) frame 0
위의 코드를 써주면 해당 frame으로 이동할 수 있음

⇒ 이때 알고 싶은 변수가 있다면 아래와 같이 변수 이름을 입력하여 해당 값을 확인할 수 있음

(gdb) p 포인터변수이름
(예시) p cfgInfo->preamble_length

THANK YOU!