# px4 빌드 시스템 분석

시작은 px4 홈페이지의 어플리케이션 작성에서 역추적해 들어갔다. 어플리케이션의 생성은 [어플리케이션 작성하기](https://dev.px4.io/kr/tutorials/tutorial_hello_sky.html) 페이지를 참조해라. 해당 링크의 설명을 보면,

1. sh git submodule init git submodule update --recursive
2. sh make px4fmu-v2\_default

의 두 단계로 이루어진다. 첫번째 명령은 추가적인 submodule을 설치하는 것이고, 두번째가 make를 이용하여 빌드를 한다. 빌드 시스템을 이해하기 위해서는 두 번째 명령부터 역추적해야 할 것이다.

## 개요

(참고 :[Firmware Download](https://github.com/px4/Firmware) )

역추적한 결과 전체적인 내용은 이러하다.

1. make명령어로 인자(빌드할 펌웨어 종류)를 받는다./Firmware/Makefile에 명시된 내용으로 실행된다. 넣어줄 인자는 px4fmu-v2\_default 이다.
2. Makefile내에서 cmake명령어를 실행시킨다. /Firmware/CMakeList.txt에 지정된 내용으로 출력된다.

* cmake /home/nnn/바탕화면/src/Firmware -G"Ninja" -DCONFIG=nuttx\_px4fmu-v2\_default형식으로 인자값이 주어진다.

1. Ninja를 이용해 빌드한다. ninja는 빌드 도구일 뿐이므로 2.까지만 분석하면 될 것이다.

## MAKEFILE

간단한 내용은 코드 내에 한글로 주석(//을 이용해 표시)을 달았다.

cmake명령이 실행되는 부분만 아래에 따로 서술한다.

메이크파일은

타겟 : 의존성

​ 커맨드

로 구성된다. 인자를 통해 타겟을 받고(참고 : 디폴트 타겟 all) 의존성이 있을경우 해당하는 의존성의 타겟을 검색하여 실행한다. (스위치문과 유사하며, 의존성을 검사하는 것이 다를 뿐이다.) 간단한 사용법만 숙지하면 되므로 구글링을 통해 익히자.

ifeq ($(wildcard .git),)  
 $(error YOU HAVE TO USE GIT TO DOWNLOAD THIS REPOSITORY. ABORTING.)  
endif  
  
# Help  
# --------------------------------------------------------------------  
# Don't be afraid of this makefile, it is just passing  
# arguments to cmake to allow us to keep the wiki pages etc.  
# that describe how to build the px4 firmware  
# the same even when using cmake instead of make.  
#  
# Example usage:  
#  
# make px4fmu-v2\_default (builds)  
# make px4fmu-v2\_default upload (builds and uploads)  
# make px4fmu-v2\_default test (builds and tests)  
#  
# This tells cmake to build the nuttx px4fmu-v2 default config in the  
# directory build/nuttx\_px4fmu-v2\_default and then call make  
# in that directory with the target upload.  
  
# explicity set default build target  
all: posix\_sitl\_default //디폴트로 실행되는 타겟. 우리는 인자를 넣으므로 무시.  
  
# Parsing  
# --------------------------------------------------------------------  
# assume 1st argument passed is the main target, the  
# rest are arguments to pass to the makefile generated  
# by cmake in the subdirectory  
FIRST\_ARG := $(firstword $(MAKECMDGOALS)) //FIRST\_ARG : px4fmu-v2\_default  
ARGS := $(wordlist 2,$(words $(MAKECMDGOALS)),$(MAKECMDGOALS)) //ARGS : NULL  
j ?= 4  
  
NINJA\_BIN := ninja //NINJA\_BIN : ninja  
ifndef NO\_NINJA\_BUILD //35~62줄까지는 빌드 도구인 ninja의 초기회라보면 된다.  
 NINJA\_BUILD := $(shell $(NINJA\_BIN) --version 2>/dev/null)  
  
 ifndef NINJA\_BUILD  
 NINJA\_BIN := ninja-build  
 NINJA\_BUILD := $(shell $(NINJA\_BIN) --version 2>/dev/null)  
 endif  
endif  
  
ifdef NINJA\_BUILD  
 PX4\_CMAKE\_GENERATOR := Ninja  
 PX4\_MAKE := $(NINJA\_BIN)  
  
 ifdef VERBOSE  
 PX4\_MAKE\_ARGS := -v  
 else  
 PX4\_MAKE\_ARGS :=  
 endif  
else  
 ifdef SYSTEMROOT  
 # Windows  
 PX4\_CMAKE\_GENERATOR := "MSYS\ Makefiles"  
 else  
 PX4\_CMAKE\_GENERATOR := "Unix\ Makefiles"  
 endif  
 PX4\_MAKE = $(MAKE)  
 PX4\_MAKE\_ARGS = -j$(j) --no-print-directory  
endif  
  
SRC\_DIR := $(shell dirname $(realpath $(lastword $(MAKEFILE\_LIST)))) //SRC\_DIR : ~/Firmware  
  
# check if replay env variable is set & set build dir accordingly  
ifdef replay //해당 없음. 실행되지 않음  
 BUILD\_DIR\_SUFFIX := \_replay  
else  
 BUILD\_DIR\_SUFFIX :=  
endif  
  
# additional config parameters passed to cmake  
ifdef EXTERNAL\_MODULES\_LOCATION //해당 없음. 실행되지 않음  
 CMAKE\_ARGS += -DEXTERNAL\_MODULES\_LOCATION:STRING=$(EXTERNAL\_MODULES\_LOCATION)  
endif  
  
ifdef PX4\_CMAKE\_BUILD\_TYPE //해당 없음. 실행되지 않음  
 CMAKE\_ARGS += -DCMAKE\_BUILD\_TYPE=${PX4\_CMAKE\_BUILD\_TYPE}  
endif  
  
# Functions  
# --------------------------------------------------------------------  
# describe how to build a cmake config  
define cmake-build //Makefile 함수 정의, cmake-build를 정의한다. 아래 상세히 설명하겠다.  
+@$(eval BUILD\_DIR = $(SRC\_DIR)/build/$@$(BUILD\_DIR\_SUFFIX))  
+@if [ $(PX4\_CMAKE\_GENERATOR) = "Ninja" ] && [ -e $(BUILD\_DIR)/Makefile ]; then rm -rf $(BUILD\_DIR); fi  
+@if [ ! -e $(BUILD\_DIR)/CMakeCache.txt ]; then mkdir -p $(BUILD\_DIR) && cd $(BUILD\_DIR) && cmake $(2) -G"$(PX4\_CMAKE\_GENERATOR)" $(CMAKE\_ARGS) -DCONFIG=$(1) || (rm -rf $(BUILD\_DIR)); fi  
+@(cd $(BUILD\_DIR) && $(PX4\_MAKE) $(PX4\_MAKE\_ARGS) $(ARGS))  
endef  
  
COLOR\_BLUE = \033[0;34m //92~98줄 단순설정  
NO\_COLOR = \033[m  
  
define colorecho  
+@echo "${COLOR\_BLUE}${1} ${NO\_COLOR}"  
endef  
  
# Get a list of all config targets cmake/configs/\*.cmake //빌드할 수 있는 펌웨어의 종류를   
ALL\_CONFIG\_TARGETS := $(basename $(shell find "$(SRC\_DIR)/cmake/configs" -maxdepth 1 ! -name '\*\_common\*' ! -name '\*\_sdflight\_\*' -name '\*.cmake' -print | sed -e 's:^.\*/::' | sort))  
# Strip off leading nuttx\_  
NUTTX\_CONFIG\_TARGETS := $(patsubst nuttx\_%,%,$(filter nuttx\_%,$(ALL\_CONFIG\_TARGETS)))  
  
# ADD CONFIGS HERE  
# --------------------------------------------------------------------  
# Do not put any spaces between function arguments.  
  
# All targets.  
$(ALL\_CONFIG\_TARGETS): //해당 없음. 무시된다.  
 $(call cmake-build,$@,$(SRC\_DIR))  
  
# Abbreviated config targets.  
  
# nuttx\_ is left off by default; provide a rule to allow that.  
$(NUTTX\_CONFIG\_TARGETS): //우리가 넣은 인자값에 해당하는 타겟!  
 $(call cmake-build,nuttx\_$@,$(SRC\_DIR)) //위에 정의한 cmake-build를 실행한다. 아래 상세히 기술.  
  
나머지는 생략!

### 115~116줄에서,

$(NUTTX\_CONFIG\_TARGETS) = aerocore2\_default aerofc-v1\_default aerofc-v1\_rtps auav-x21\_default crazyflie\_default esc35-v1\_default mindpx-v2\_default nxphlite-v3\_default px4-same70xplained-v1\_default px4-stm32f4discovery\_default px4cannode-v1\_default px4esc-v1\_default **px4fmu-v2\_default** px4fmu-v2\_lpe px4fmu-v2\_test px4fmu-v3\_default px4fmu-v3\_rtps px4fmu-v4\_default px4fmu-v4\_rtps px4fmu-v4pro\_default px4fmu-v4pro\_rtps px4fmu-v5\_default px4fmu-v5\_rtps px4io-v2\_default px4nucleoF767ZI-v1\_default s2740vc-v1\_default tap-v1\_default

이다. 따라서 우리가 넣은 인자 make px4fmu-v2\_default와 매치되고 해당하는 커맨드(116줄)가 실행된다

$(call cmake-build,nuttx\_$@,$(SRC\_DIR))

call 에 의해 위에 정의된 cmake-build함수가 해당하는 인자로 실행된다.

첫 번째 인자는 nuttx\_$@ 에 의해nuttx\_px4fmu-v2\_default이다. ($@는 타겟을 의미한다 즉 px4fmu-v2\_default이다.)

두 번째 인자는 $(SRC\_DIR) = /firmware이다.

### 85~90줄 cmake-build함수

115~116줄에서 위에서 설명된 인자로 호출된다.

define cmake-build //Makefile 함수 정의, cmake-build를 정의한다. 아래 상세히 설명하겠다.  
+@$(eval BUILD\_DIR = $(SRC\_DIR)/build/$@$(BUILD\_DIR\_SUFFIX))  
+@if [ $(PX4\_CMAKE\_GENERATOR) = "Ninja" ] && [ -e $(BUILD\_DIR)/Makefile ]; then rm -rf $(BUILD\_DIR); fi  
+@if [ ! -e $(BUILD\_DIR)/CMakeCache.txt ]; then mkdir -p $(BUILD\_DIR) && cd $(BUILD\_DIR) && cmake $(2) -G"$(PX4\_CMAKE\_GENERATOR)" $(CMAKE\_ARGS) -DCONFIG=$(1) || (rm -rf $(BUILD\_DIR)); fi  
+@(cd $(BUILD\_DIR) && $(PX4\_MAKE) $(PX4\_MAKE\_ARGS) $(ARGS))  
endef

2줄 : BUILD\_DIR을 정의한다. 정의되는 값은 /firmware/build/px4fmu-v2\_default

3줄 : 빌드도구 ninja가 있고, 그리고, 빌드 디렉터리에 makefile이 있으면 디렉터리를 지워서 깨끗이 한다.

4줄 : 빌드 디렉터리에 CMakeCache.txt가 있는지 확인한다. 있다면 이전에 이미 cmake가 실행되었으므로 ninja를 실행할 준비가 된 것이다. 없다면 ninja를 실행하기 위해 cmake명령을 실행한다. 해당하는 명령은 cmake /home/nnn/바탕화면/src/Firmware -G"Ninja" -DCONFIG=nuttx\_px4fmu-v2\_default이다.($(1)과 $(2)는 함수에 전달된 첫 번째 두 번째 인자를 의미한다.)

5줄 : 빌드 디렉터리에 이동하여 ninja를 이용해 빌드한다.

결과로 px4fmu-v2\_default.px4가 생성된다.

## 증명

Makefile을 분석해 얻은 명렁들로 쉘에서 실행하겠다. 펌웨어가 생성된다면, 분석이 맞은 것이다.

아래 명령을 통해 필요한 소스를 다운받는다.

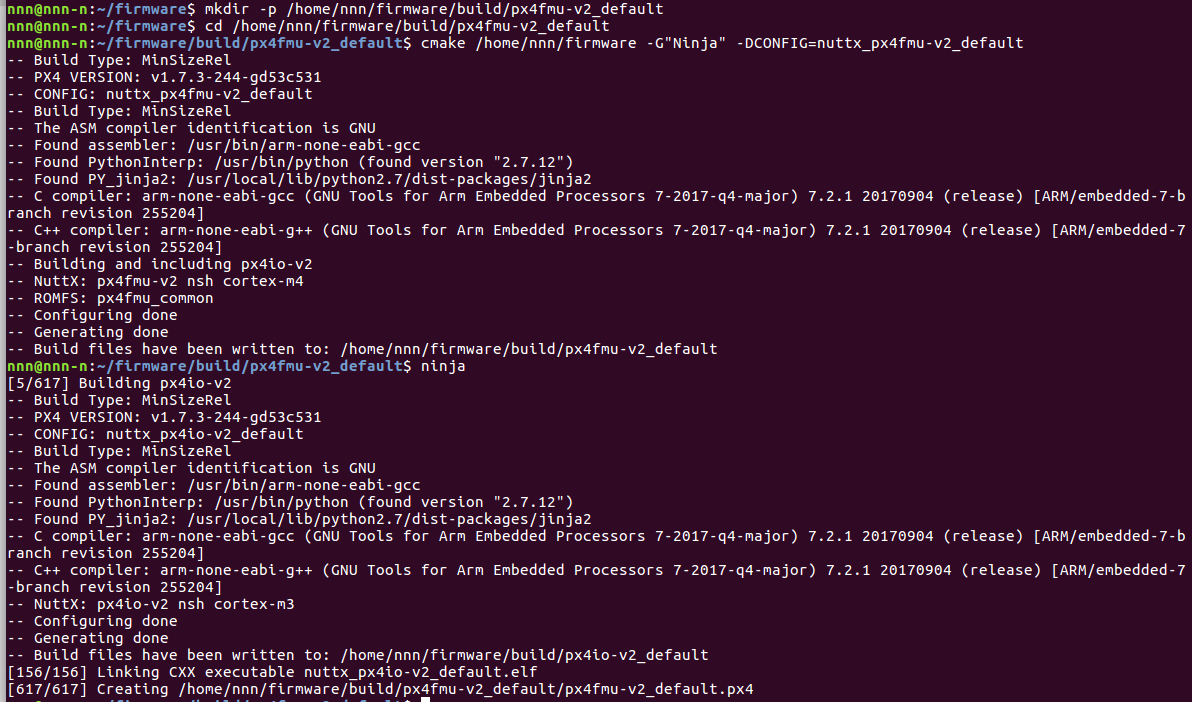
$cd ~  
$git clone https://github.com/px4/firmware  
$cd firmware/  
$git submodule init  
$git submodule update --recursive

초기 상황이므로 /build/px4fmu-v2\_default 디렉터리와 그 하위의 Makefile CMakeCache.txt 도 존재하지 않을 것이다. 따라서cmake-build의 4째줄 5째줄만 실행된다. 해당하는 명령은

nnn@nnn-n:~/firmware$ mkdir -p /home/nnn/firmware/build/px4fmu-v2\_default  
nnn@nnn-n:~/firmware$ cd /home/nnn/firmware/build/px4fmu-v2\_default  
nnn@nnn-n:~/firmware/build/px4fmu-v2\_default$ cmake /home/nnn/firmware -G"Ninja" -DCONFIG=nuttx\_px4fmu-v2\_default  
nnn@nnn-n:~/firmware/build/px4fmu-v2\_default$ ninja

이다.

결과는



Alt text

빌드 성공.

### 참고

define cmake-build  
+@$(eval BUILD\_DIR = $(SRC\_DIR)/build/$@$(BUILD\_DIR\_SUFFIX))  
+@if [ $(PX4\_CMAKE\_GENERATOR) = "Ninja" ] && [ -e $(BUILD\_DIR)/Makefile ]; then rm -rf $(BUILD\_DIR); fi  
+@if [ ! -e $(BUILD\_DIR)/CMakeCache.txt ]; then mkdir -p $(BUILD\_DIR) && cd $(BUILD\_DIR) && cmake $(2) -G"$(PX4\_CMAKE\_GENERATOR)" $(CMAKE\_ARGS) -DCONFIG=$(1) || (rm -rf $(BUILD\_DIR)); fi  
+@(cd $(BUILD\_DIR) && $(PX4\_MAKE) $(PX4\_MAKE\_ARGS) $(ARGS))  
@echo "value print"  
@echo "FIRST\_ARG : $(FIRST\_ARG)"  
@echo "ARGS : $(ARGS) "  
@echo "NINJA\_BIN : $(NINJA\_BIN) "  
@echo "NINJA\_BUILD : $(NINJA\_BUILD) "  
@echo "PX4\_CMAKE\_GENERATOR : $(PX4\_CMAKE\_GENERATOR) "  
@echo "PX4\_MAKE : $(PX4\_MAKE) "  
@echo "PX4\_MAKE\_ARGS : $(PX4\_MAKE\_ARGS) "  
@echo "PX4\_CMAKE\_BUILD\_TYPE : $(PX4\_CMAKE\_BUILD\_TYPE) "  
@echo "BUILD\_DIR\_SUFFIX : $(BUILD\_DIR\_SUFFIX) "  
@echo "CMAKE\_ARGS : $(CMAKE\_ARGS) "  
@echo "BUILD\_DIR : $(BUILD\_DIR) "  
@echo "ALL\_CONFIG\_TARGETS : $(ALL\_CONFIG\_TARGETS) "  
@echo "NUTTX\_CONFIG\_TARGETS : $(NUTTX\_CONFIG\_TARGETS) "  
@echo "SRC\_DIR : $(SRC\_DIR) "  
@echo "BUILD\_DIR : $(BUILD\_DIR) "  
endef

위와 같이 cmake-build를 수정함으로써 px4fmu-v2\_defauㅣt를 빌드할 때의 변수를 알 수 있다. 사용법은 $make px4fmu-v2\_default 라고 쉘에 입력하면 된다.

## CMakeList.txt

이제 Makefile이 실행한 cmake명령어가 어떻게 빌드할지를 정하는지 알아보자. 해당 내용은 /firmware/CMakeList.txt에 작성되어 있다.

해당 파일에는 add\_subdirectory(source\_dir)가 있는데, 이는 소스코드의 경로를 지정하는 함수이다. 지정된 경로에는 그 소스코드에 해당하는 CMakeList.txt가 있으며, 이 파일을 통해 소스코드가 어떤 플래그와 설정을 가지고 컴파일 되는지 알 수 있다.

add\_subdirectory(source\_dir)가 지정하는 디렉터리를 message()를 이용해 모두 출력하였다. 아래와 같은 방법이다

예시

foreach(driver ${config\_df\_driver\_list})  
 add\_subdirectory(src/lib/DriverFramework/drivers/${driver})  
 message("src/lib/DriverFramework/drivers/${driver}") //추가된 함수  
endforeach()

endforeach()

결과는

src/drivers/differential\_pressure  
src/drivers/distance\_sensor  
src/drivers/airspeed  
src/drivers/barometer/ms5611  
src/drivers/boards  
src/drivers/camera\_trigger  
src/drivers/device  
src/drivers/gps  
src/drivers/l3gd20  
src/drivers/led  
src/drivers/lsm303d  
src/drivers/magnetometer/hmc5883  
src/drivers/magnetometer/lis3mdl  
src/drivers/mpu6000  
src/drivers/mpu9250  
src/drivers/pwm\_input  
src/drivers/pwm\_out\_sim  
src/drivers/px4flow  
src/drivers/px4fmu  
src/drivers/px4io  
src/drivers/rgbled  
src/drivers/stm32  
src/drivers/stm32/adc  
src/drivers/stm32/tone\_alarm  
src/drivers/vmount  
src/modules/sensors  
src/systemcmds/hardfault\_log  
src/systemcmds/mixer  
src/systemcmds/mtd  
src/systemcmds/param  
src/systemcmds/perf  
src/systemcmds/pwm  
src/systemcmds/reboot  
src/systemcmds/top  
src/systemcmds/ver  
src/modules/camera\_feedback  
src/modules/commander  
src/modules/events  
src/modules/land\_detector  
src/modules/load\_mon  
src/modules/mavlink  
src/modules/navigator  
src/modules/ekf2  
src/modules/fw\_att\_control  
src/modules/fw\_pos\_control\_l1  
src/modules/mc\_att\_control  
src/modules/mc\_pos\_control  
src/modules/vtol\_att\_control  
src/modules/logger  
src/modules/dataman  
src/modules/systemlib  
src/modules/systemlib/param  
src/modules/uORB  
src/lib/controllib  
src/lib/conversion  
src/lib/DriverFramework/framework  
src/lib/ecl  
src/lib/geo  
src/lib/geo\_lookup  
src/lib/launchdetection  
src/lib/led  
src/lib/mathlib  
src/lib/mathlib/math/filter  
src/lib/mixer  
src/lib/runway\_takeoff  
src/lib/tailsitter\_recovery  
src/lib/tunes  
src/lib/version  
src/platforms/common  
src/platforms/nuttx  
src/platforms/nuttx/px4\_layer  
platforms/nuttx

이다.

이 문서는 아직 미완성 입니다. 위 경로의 CMakeList.txt를 분석하고, /firmware/CMakeList.txt를 마저 분석하면 nuttx와 모듈들이 어떻게 빌드되는지 이해할 수 있다고 생각됩니다.

현재 firmware/cmake/common/ 의 라이브러리 함수를 통해 모듈이 빌드되는 윤곽을 잡았습니다.

firmware/platforms/nuttx/CMakeLists.txt 에 nuttx가 어떻게 빌드되는지 명시되어 있지만 분석에 애를 먹고 있습니다.

그리고 빌드된 nuttx와 모듈들이 어떻게 링크 되는지도 분석 중에 있습니다.