PX4 on H7

Roy Ratcliffe[1,*]

Abstract

PX4 Autopilot firmware project offers a sophisticated system for drone flight management. This article explores the major bumps and bends that a developer might first encounter when launching a project based on PX4. The development platform is Windows 11 running Windows Subsystem for Linux 2 running Ubuntu Jammy. For evaluation, the article troubleshoots PX4 using a STM32H747I-DISCO evaluation board by ST.

What is PX4? At its most abstract, PX4 Autopilot is an open-source multi-architecture tool for developing drone flight management firmware.

1. Discovery

ST's STM32 H747I Discovery is an evaluation board based on the STM32H747xI dual-core embedded system-on-chip. The H747 series sandwiches an ARM Cortex M7 and M4 with a plethora of on-chip peripherals. NuttX carries support for this board 'out of the box' although it does *not* utilise the standard ST-manufacturer's hardware abstraction layer (HAL) but instead deploys its own NuttX µHAL.

Setting it up has a few "gotcha" moments, however. The following sections describe tips, tricks and other useful lessons learnt while developing a basic BSP for the H7; this is just a smaller subset of the entire corpus of tricks but serves as the highlights.

1.1. ARM Toolchain Series

The PX4 Autopilot firmware is a CMake project with specific toolchain version requirements. Successful and *noiseless* builds require an older version of Kitware's CMake, version 3.16.2. Later versions trigger a policy warning about Python package macros. The ARM compiler toolchain similarly fires warnings and errors with the latest versions[2]. The firmware utilises 9-series[3] ARM compiler version 9.3.1 for canonical compilation.

Later versions of the toolchain separate standard library headers for example; hence the compiler does not supply the standard `math.h` header. Smoother for development if the build picks up a more compatible version of the compiler toolchain until the project progresses to allow for the use of more up-to-date toolchain versions without breakages.

1.2. Board Support Package (BSP)

A minimal BSP lays out within the project as follows.

```
boards/stm32/h747i-disco/
  default.px4board
  nuttx-config
     include
     nsh
        defconfig
     src
  src
     CMakeLists.txt
```

Files with extension `px4board` define the board. The build system searches for these in order to find all available boards to configure and build.

At a minimum, before configuring the board, the default NuttX configuration needs to know the architecture and the location of the custom-board directory.

```
CONFIG_ARCH="arm"
CONFIG_ARCH_BOARD_CUSTOM=y
CONFIG_ARCH_BOARD_CUSTOM_DIR="../../../../boards/stm32/h747i-disco/nuttx-config"
CONFIG_ARCH_BOARD_CUSTOM_DIR_RELPATH=y
```

Running the text-based menu configuration tool now works. The build system fails *without* the ARM architecture configuration. Circular requirements exist: configuration depends on the architecture; the build tooling for configuration needs to know the architecture before configuration. The developer must first 'bootstrap' the build with at least the general architecture, ARM architecture in this case. The following `make` commands succeed by launching menu-driven configuration tools.

[*] Corresponding author
Email address: `roy@ratcliffe.me` (Roy Ratcliffe)

[1] See more at GitHub.

[2] Missing `math.h` and `fabsf` for example.

[3] ARM publish the patched 9-series as 9-2020-q2-update corresponding to 9.3.1. Install it by downloading the tarball and unzipping it then applying `sudo tar -C /opt -xjf` to the tarball. Append the installation's `bin` directory to the search path.

```
make stm32_h747i-disco menuconfig
make stm32_h747i-disco boardconfig
```

Target `menuconfig` launches the NuttX configurator while `boardconfig` launches the PX4 board configurator. See Figure 1 and Figure 2.

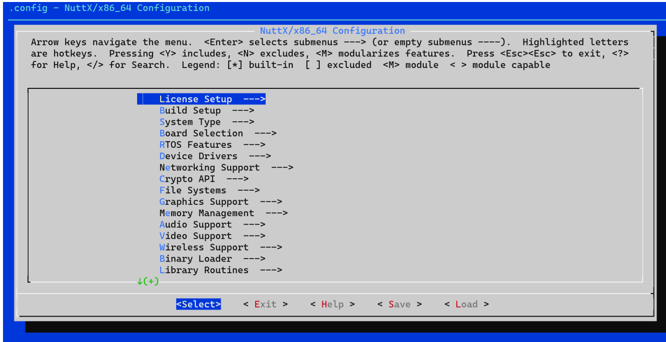

Figure 1: NuttX configuration menu

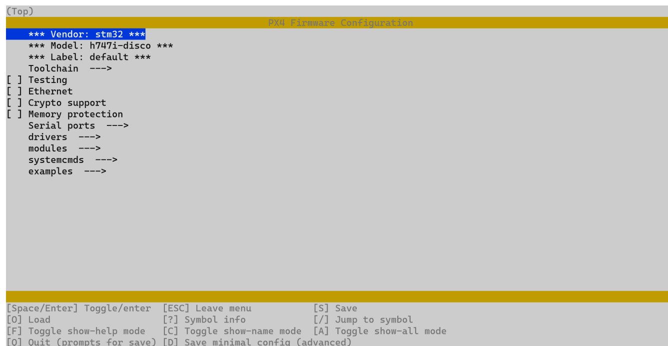

Figure 2: PX4 board configuration

1.3. Version Tag

The build system needs to see Git tags of the form $v_{major.minor.patch} - dev$ otherwise it fails. Pull the remote tags if not already pulled.

1.4. Board Header

Builds always fail with various errors without three **essential** `board_config.h` header includes:

```c
#include <stm32.h>
#include <px4_platform_common/px4_config.h>
#include <px4_platform_common/board_common.h>
```

All sorts of issues arise without including the PX4 configuration and the common board header—too many to describe in any detail—suffice it to say that these included headers act as bridges between the various PX4 sub-layers. Without them appearing in the board configuration, various PX4 platform-oriented components and µHAL dependencies collide with sundry unresolved enumerations, defines and function prototypes. The board configuration header glues the disparate layer of software together.

1.5. Board-Level Drivers

Linking the firmware fails without two important libraries:

```
ld: cannot find -lromfs
ld: cannot find -ldrivers_board
```

NuttX needs configuring with ROMFS filesystem support in order to resolve the `romfs` dependency.

"Drivers board" library refers to drivers for the board customisation. The sources already exist but within the NuttX 'platform boards' *not* within the PX4 upper layer. They belong to the PX4 NuttX repository at GitHub. The build system does not automatically find these sources and set them up for compiling and linking with the firmware. They need to be picked out and added to the `drivers_board` library.

These out-of-the-box board drivers compile with errors, however. The compiler complains about the comparison of integer expressions of different signedness: 'int' and 'unsigned int'[4].

```c
for (i = 0; i < ARRAYSIZE(g_ledmap); i++)
```

This needs correcting but the fix lives in a Git submodule as part of the NuttX repository. The proper fix would require a successful pull request applied to that repository. Tempting to create one. As an interim workaround, the custom drivers can copy and patch the offending sources.

1.6. Serial Ports

Compiling fails without at least *one* UART equipped at the NuttX layer. Just configure one using the NuttX configuration tool.

```
CONFIG_STM32H7_USART1=y
```

NuttX equips serial-driver support by default but it does *not* enable any UART devices. The compiler sees an error.

1.7. Firmware Prototype

As a final version-stamping step, the PX4 build system converts the linked ELF to PX4 format using its custom `px_mkfw` tool. This step requires a `firmware.prototype` JSON file that describes the board assigning a board identifier together with other summary information.

```json
{
    "board_id": 999,
    "magic": "PX4FWv0",
    "description": "Firmware for STM32H747I Discovery",
    "image": "",
```

[4] `-Werror=sign-compare`

```
    "build_time": 0,
    "summary": "STM32H747I-DISCO",
    "version": "0.1",
    "image_size": 0,
    "image_maxsize": 2080768,
    "git_identity": "",
    "board_revision": 0
}
```

1.8. General Tricks

Cleaning up the repo clone and its sub-modules to restart the firmware cleanly using `git clean`.

```
git clean -fdx
git submodule foreach --recursive git clean -fdx
```

Update all the submodules using:

```
make submodulesupdate
```

2. Results

The result is a *minimal* set of firmware for running PX4 autopilot on the H747I Discovery evaluation board by reusing the existing NuttX configuration and source files: ideal for evaluating a design without pre-loaded features standing in the way.

```
[373/375] Linking CXX executable stm32_h747i-disco_default.elf
Memory region         Used Size  Region Size  %age Used
           itcm:           0 GB        64 KB      0.00%
          flash:        22452 B         2 MB      1.07%
          dtcm1:           0 GB        64 KB      0.00%
          dtcm2:           0 GB        64 KB      0.00%
           sram:         2288 B       512 KB      0.44%
          sram1:           0 GB       128 KB      0.00%
          sram2:           0 GB       128 KB      0.00%
          sram3:           0 GB        32 KB      0.00%
          sram4:           0 GB        64 KB      0.00%
          bbram:           0 GB         4 KB      0.00%
```

It also flashes, runs and debugs. PX4 kindly sets up a launch configuration for VS Code. See Figure 3. Debugger connection on a Windows machine running Windows Subsystem for Linux uses `usbipd` for bridging USB to WSL. The vendor-product identifier `0483:374e` corresponds to the evaluation board's ST-Link in the 'attach' command snippet below.

```
usbipd wsl attach -i 0483:374e -d Ubuntu
```

3. Conclusions

The PX4 firmware configures itself using a deeply-nested chain of symbolically-linked CMake, Make sub-projects and shell scripts which manifest at the compiler stages as a complicated graph of interdependent pre-processor manifest constants that enable or disable component-level features. It works by building a set of NuttX operating-system static libraries and then linking them against the board-level compiled objects. This approach works well when no compile-time issues exist but is difficult to debug when errors arise—as inevitably they do when porting to new platforms.

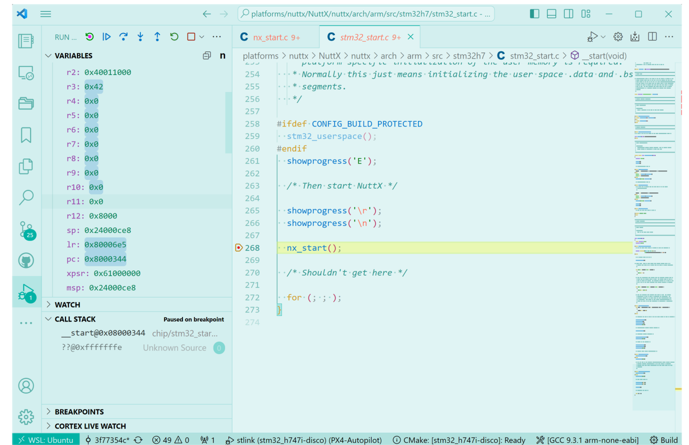

Figure 3: VS Code debugger

It takes some time and effort to navigate PX4. It comprises a number of sophisticated interconnected layers: the NuttX operating system, µHAL, board support layer, PX4 drivers, modules, and applications. This is to be expected for such a feature-rich and capable body of software. Starting a project from a clean working slate allows for an uncluttered approach.