Built on NuttX

Flown on Drones

http://www.nscdg.com

Gouda, NL 07/17/2019
PX4 is a BSD licensed Open Source Autopilot

Website: https://px4.io  
Github: https://github.com/PX4

The PX4 project was started by Lorenz Meier in 2008 on and flown on Pixhawk

Second generation Pixhawk drone – Zurich 2009

“A decade ago, little did I know that my student project at the Computer Vision and Geometry Lab at ETH Zurich would end up becoming the de facto standard in the drone industry.” - Lorenz Meier
Pixhawk is an Open Hardware Reference Standard

Website: http://pixhawk.org

Github: https://github.com/pixhawk

Open Hardware for Autonomous Aviation

- **FMUv1** Pixhawk
  - STM32F407/STM32F100

- **FMUv2** Pixhawk 1
  - STM32F429/STM32F100

- **FMUv3** Pixhawk 2
  - STM32F429/STM32F100

- **FMUv4** Pixracer
  - STM32F427

- **FMUv4** Pixhawk 3 Pro
  - STM32F469/STM32F100
What is an Open Hardware Reference Standard?

Website: https://dev.px4.io/v1.9.0/en/debug/reference-design.html
What Happens when you create an Open Hardware Reference Standard?
What Happens when you create an Open Hardware Reference Standard?
Pixhawk FMUv{5:6}[X] Reference Standard

Current and Future FMU Versions

<table>
<thead>
<tr>
<th>FMUv5 Pixhawk 4</th>
<th>FMUv5X Pixhawk 5X</th>
<th>FMUv6 Pixhawk 6</th>
<th>FMUv6X Pixhawk 6X</th>
</tr>
</thead>
<tbody>
<tr>
<td>STM32F765/STM32F100</td>
<td>STM32F765/STM32F100</td>
<td>STM32H753/STM32F100</td>
<td>STM32H753/STM32F100</td>
</tr>
</tbody>
</table>

Prototype phase
Why PX4 Chose NuttX

- The BSD Licensing - "BSD licenses are a family of permissive free software licenses, imposing minimal restrictions on the use and distribution of covered software. This is in contrast to copyleft licenses, which have share-alike requirements. The original BSD license was used for its namesake, the Berkeley Software Distribution (BSD)"

- "The Portable Operating System Interface (POSIX) is an IEEE standard that helps compatibility and portability between operating systems. Theoretically, POSIX compliant source code should be seamlessly portable. In the real world, application transition often runs into system specific issues"

- Real Time OS

- The scalability and degree of freedom to which it can be modified to suit application specific needs, from small footprint to large.

- Code Quality and conformity.
How is PX4 Built on NuttX

- PX4 drives the NuttX Makefile build system using make and cmake.
  - It is an out of tree build
  - We used to use NuttX make export
    - We now build the NuttX libraries as cmake projects.
- PX4 uses Cmake
  - CMake is an extensible, open-source system that manages the build process in an operating system and in a compiler-independent manner.
- PX4 uses ccache
  - ccache is a compiler cache. It speeds up recompilation by caching the result of previous compilations and detecting when the same compilation is being done again.
- PX4 uses ninja[build]
  - Ninja is a small build system with a focus on speed. It differs from other build systems in two major respects: it is designed to have its input files generated by a higher-level build system, and it is designed to run builds as fast as possible.
PX4 on NuttX

It looks like Make on the command line

- `make help` - list all targets
- `make px4_fmu-v5` - build PX4 for fmuv5 hardware
- `make nxp_fmurt1062-v1` - build PX4 for NXP 1060 RT hardware

**Familiar but different**

- `make px4_fmu-v5 oldconfig`
- `make px4_fmu-v5 menuconfig`

- We build what we can in parallel
- We drive the defconfig to .config process
- We dynamically add to the builtins
- We use the provided magic:
  - `CONFIG_ARCH_BOARD_CUSTOM_DIR="../nuttx-config"`
  - `CONFIG_ARCH_BOARD_CUSTOM_NAME="px4"`
PX4 on NuttX

We split the source and NuttX configuration. We build NuttX to libraries. The board library in nuttx is empty! Board source is built in PX4 and linked to the NuttX libraries.
PX4 on NuttX
PX4 on NuttX

Test as you code
35 Complete builds in < 11 Minutes

Lean heavily on CI Tools:

Build all PRs - prevents merging code breaks the build.

Run a style check - prevents merging code that is not to the coding standard
PX4 on NuttX

Test as you fly
Lean heavily on CI Tools:
PX4 on NuttX

PX4 has been working on complete CI for NuttX

20 Build configurations in < 4 minutes
12-20 Seconds Each Per build of <board>/<config>!

How can we help?
PX4 team is willing to add AND Maintain full CI on NuttX in tree.

But we need some changes to support it.
Inclusion of yaml files and cmake

Add a versioning Knot linking apps to nuttx
Some cool PX4 apps

dmesg
hardfault_log
top
uORB

Ideas for future

Fully nested prioritized interrupt structure.
Compile time Device Tree
What is a HardFault?

Within NuttX, all roads lead to up_assert via the common vector

HardFault
MemManage
BusFault
UsageFault

Common causes:
- Both software and hardware can cause HardFaults
- Hardware accessing a peripheral that is not enabled - BusFault
- Executing a pure virtual function (AKA: null pointer execution)
- Dereferencing a null pointer
- Stack crash (AKA: stack smashing) or wild pointer corrupting data used downstream
Scale of difficulty debugging a HardFault

Simple to debug:
(Repeatable occurrence of HardFault)
- Hardware accessing a peripheral that is not enabled
- Executing a pure virtual function
- Dereferencing a null pointer

Complex to debug:
(random occurrence of HardFault)
- Stack crash or wild pointer corrupting data used downstream
- Inappropriate hardware interrupt priority settings
The Evolution leading to the Pixhawk debug adapter
Tools - HardFault debugging is not as difficult as it used to be

The old days:
Bond-out InCurcuitEmulator (ICE)
$15,000 USD

Current:
JTAG debugger
$20.00 USD
Live and Postmortem Debugging

Live:

GNU ARM → GNU MCU Eclipse!

Set a breakpoint on up_hardfault and up_assert
Set the PC equal to the LR
Select assembly single step
And step to bx lr instruction in do_irq that will return you to the line of code that caused the HardFault

Postmortem:

Reviewing the HardFault log
Choosing addresses in flash
And disassembling at those addresses