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

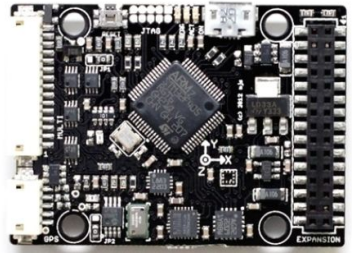
“A decade ago, little did I know that my student project at the Computer Vision and Geometry Lab at ETH Zurich would end up becoming the de facto standard in the drone industry.” - Lorenz Meier

Pixhawk is an Open Hardware Reference Standard

Website: http://pixhawk.org **Github:** https://github.com/pixhawk

Open Hardware for Autonomous Aviation

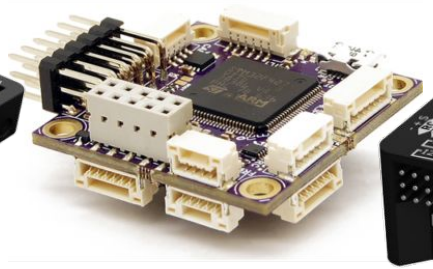
FMUv1 Pixhawk
STM32F407/STM32F100



FMUv2 Pixhawk 1
STM32F429/STM32F100

FMUv3 Pixhawk 2
STM32F429/STM32F100

FMUv4 Pixracer
STM32F427

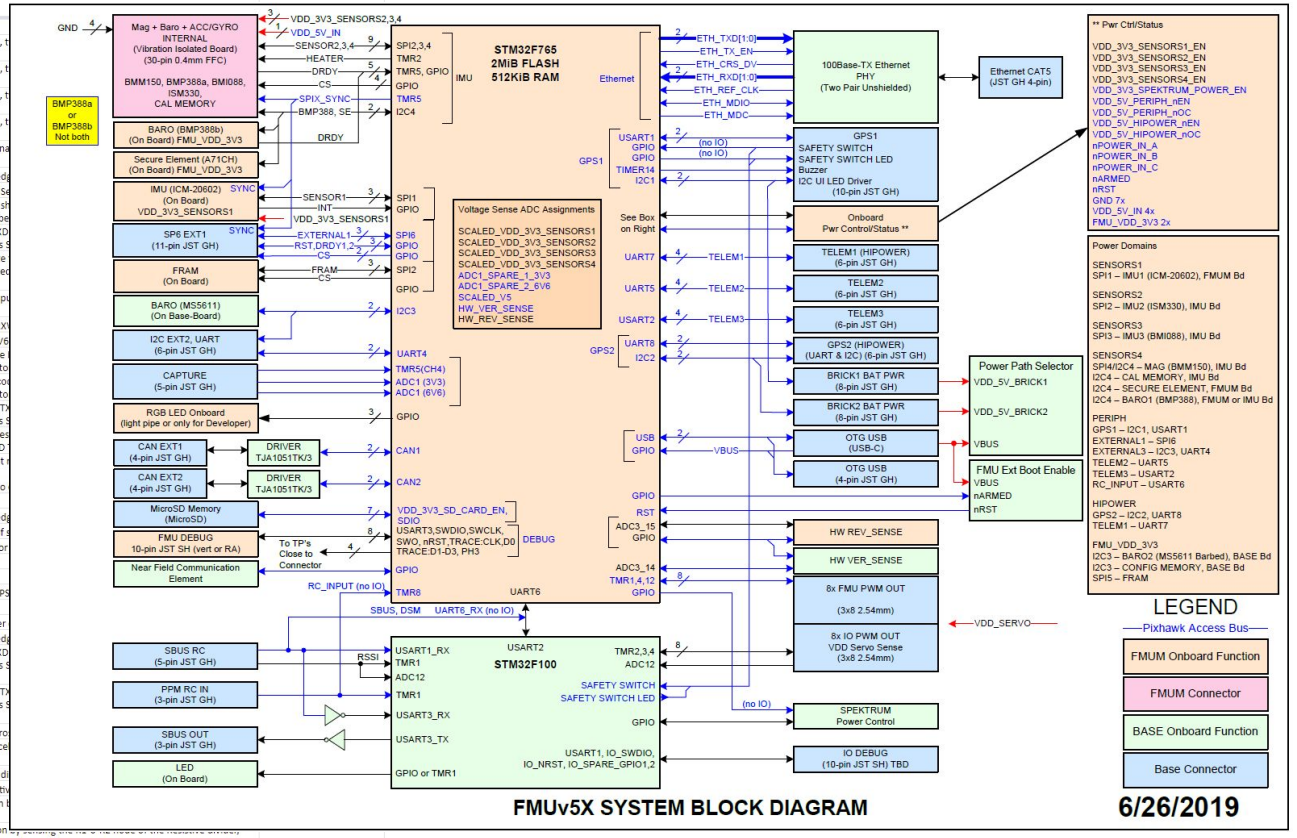


FMUv4 Pixhawk 3 Pro
STM32F469/STM32F100

What is an Open Hardware Reference Standard?

Website: https://dev.px4.io/v1.9.0/en/debug/reference-design.html

Ordinal	PORT	PIN	176-pin STM32F765IIK Signal	176-pin STM32F765IIK FMU USAGE	Functional Description
0	PA	0	ADC1_IN0	BAT1_V	Analog INPUT: Battery (aka Brick) voltage sense, t low source impedance
1	PA	1	ADC1_IN1	BAT1_I	Analog INPUT: Battery (aka Brick) current sense, t Brick module
2	PA	2	ADC1_IN2	BAT2_V	Analog INPUT: Battery (aka Brick) voltage sense, t low source impedance
3	PA	3	ADC1_IN3	BAT2_I	Analog INPUT: Battery (aka Brick) current sense, t Brick module
4	PA	4	[illegible]	ADC1_SPARE_2	Analog INPUT: Can be used to sense any additona dependent. Leave NC if not used
5	PA	5	TIM2_CH1_IN	FMU_CAP1	Digital INPUT: Typicaly used for PWM input or edg
6	PA	6	SPI1_MISO	SPI1_MISO_SENSOR1	Digital INPUT: SPI Bus 1 Master In Slave Out for Se
7	PA	7	TIM14_CH1	HEATER	Digital OUTPUT: Active High, Open Circuit state sh FET to switch on and off resistor as heater, can be
8	PA	8	CAN3_RX	CAN3_RX	Digital INPUT: Connected to CAN Transceiver RXD with Silent control (A HIGH level on pin S selects S
9	PA	9	USB_OTG_FS_VBUS	VBUS	Digital INPUT: Provides the VBUS sensing feature used to enter bootloader when FMU is connected terminator w/ ESD protection
10	PA	10	TIM1_CH3	FMU_CH2	Digital I/O: Output: PWM, GPIO, Input: PWM Inpu note
11	PA	11	USB_OTG_FS_DM	USB_DM	Digital I/O: USB D Minus, recommend NUF2042X
12	PA	12	USB_OTG_FS_DP	USB_DP	Digital I/O: USB D Plus, recommend NUF2042XV6
13	PA	13	SWDIO	JTAG-SWDIO	Digital I/O: Connected to Pin 4 of the Dronecode https://wiki.dronecode.org/workgroup/connecto
14	PA	14	SWCLK	JTAG-SWCLK	Digital INPUT: Connected to Pin 5 of the Dronecod https://wiki.dronecode.org/workgroup/connecto
15	PA	15	CAN3_TX	CAN3_TX	Digital OUTPUT: Connected to CAN Transceiver TX with Silent control (A HIGH level on pin S selects S
16	PB	0	ADC1_IN8	RSSI_IN	Analog INPUT: RSSI connected through 10K series 220R to connector pin (May Also be used as TBD
17	PB	1	TIM3_CH4	nLED_RED	Used for status, may be a discrete LED. Does not ANODE can be V5 or V3.3) Digital OUTPUT: Active Low, PWM capable pin to
18	PB	2			available Digital GPIO
19	PB	3	TIM2_CH2_IN	FMU_CAP2	Digital INPUT: Typicaly used for PWM input or edg
20	PB	4	PB4	SPI1_DRDY1_ICM20689	Digital INPUT: Ready Interrupt from ICM20689 if s
21	PB	5	SPI6_MOSI	SPI6_MOSI_EXTERNAL2	Digital OUTPUT: SPI Bus 6 Master Out Slave In for
22	PB	6	USART1_TX	USART1_TX_GPS1	Digital OUTPUT: USART1 TX is used for GPS1
23	PB	7	USART1_RX	USART1_RX_GPS1	Digital INPUT: USART1 RX is used for GPS1
24	PB	8	I2C1_SCL	I2C1_SCL_GPS1	Digital OUTPUT: I2C Bus 1's Clock paired with GPS
25	PB	9	I2C1_SDA	I2C1_SDA_GPS1	Digital I/O: I2C Bus 1's Data paired with GPS1
26	PB	10	PB10	nSPI5_RESET_EXTERNAL1	Digital OUPUT: Reserved for SPI 5 reset or power
27	PB	11	TIM2_CH4_IN	FMU_CAP3	Digital INPUT: Typicaly used for PWM input or edg
28	PB	12	CAN2_RX	CAN2_RX/UART5_RX_ESC	Digital INPUT: Connected to CAN Transceiver RXD with Silent control (A HIGH level on pin S selects S UART5 RX for Serial ESC
29	PB	13	CAN2_TX	CAN2_TX/UART5_TX_ESC	Digital OUTPUT: Connected to CAN Transceiver TX with Silent control (A HIGH level on pin S selects S UART5 TX for Serial ESC
30	PB	14	PB14	SPI1_DRDY2_BMI055_GYRO	Digital INPUT: Ready Interrupt from BMI055 Gyro
31	PB	15	PB15	SPI1_DRDY3_BMI055_ACC	Digital INPUT: Ready Interrupt from BMI055 Acce present
32	PC	0	ADC1_IN10	SCALED_V5	Analog INPUT: V5 Input voltage sense, resistive di
33	PC	1	ADC1_IN11	SCALED_VDD_3V3_SENSORS	Analog INPUT: Sensor 3.3 V voltage sense, resistiv
34	PC	2	ADC1_IN12	HW_VER_SENSE	Analog INPUT: Used to determine Board Version HW REV and VER ID tab
35	PC	3	ADC1_IN13	HW_REV_SENSE	Analog INPUT: Used to determine Board Revision see HW REV and VER ID tab



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

Prototype phase

FMUv5 Pixhawk 4
STM32F765/STM32F100

FMUv5X Pixhawk 5X
STM32F765/STM32F100

FMUv6 Pixhawk 6
STM32H753/STM32F100

FMUv6X Pixhawk 6X
STM32H753/STM32F100

PX4 on NuttX

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.

PX4 on NuttX

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 nija[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
```

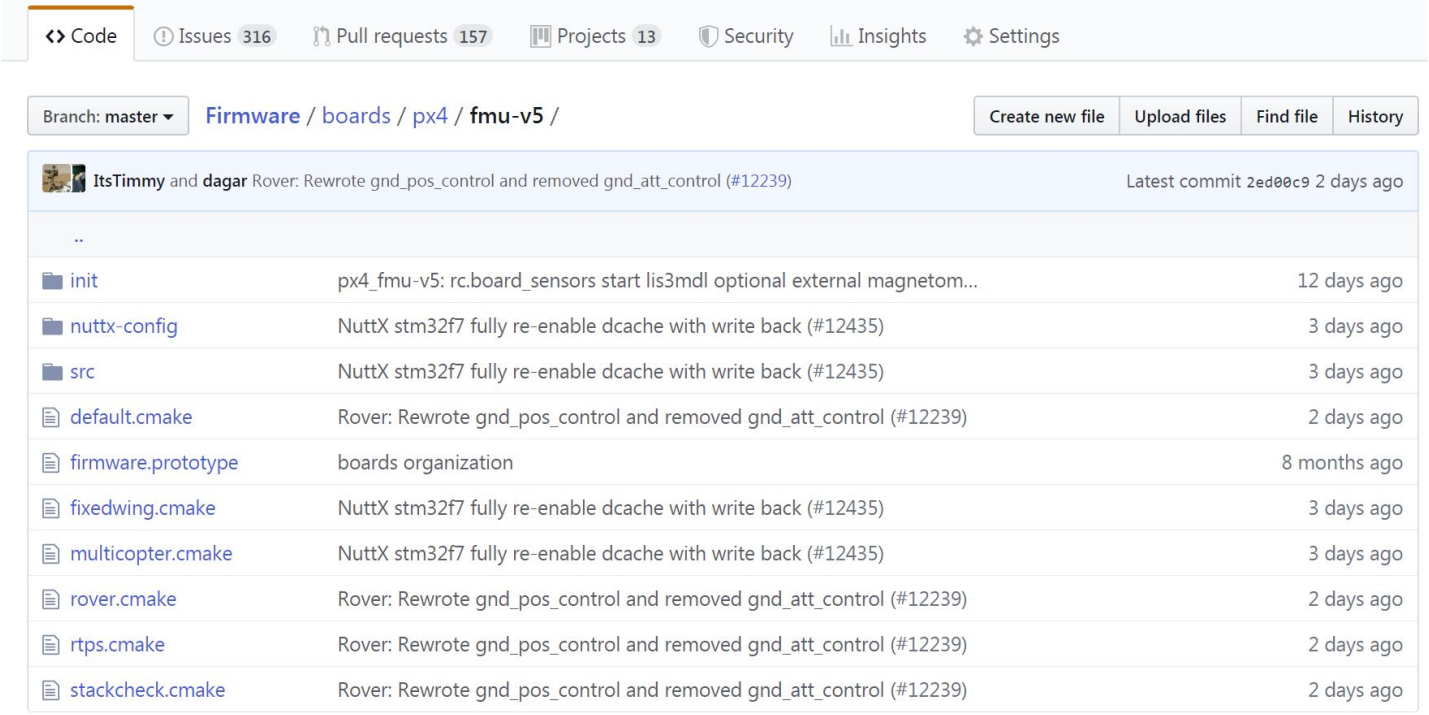
Familiar but different

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

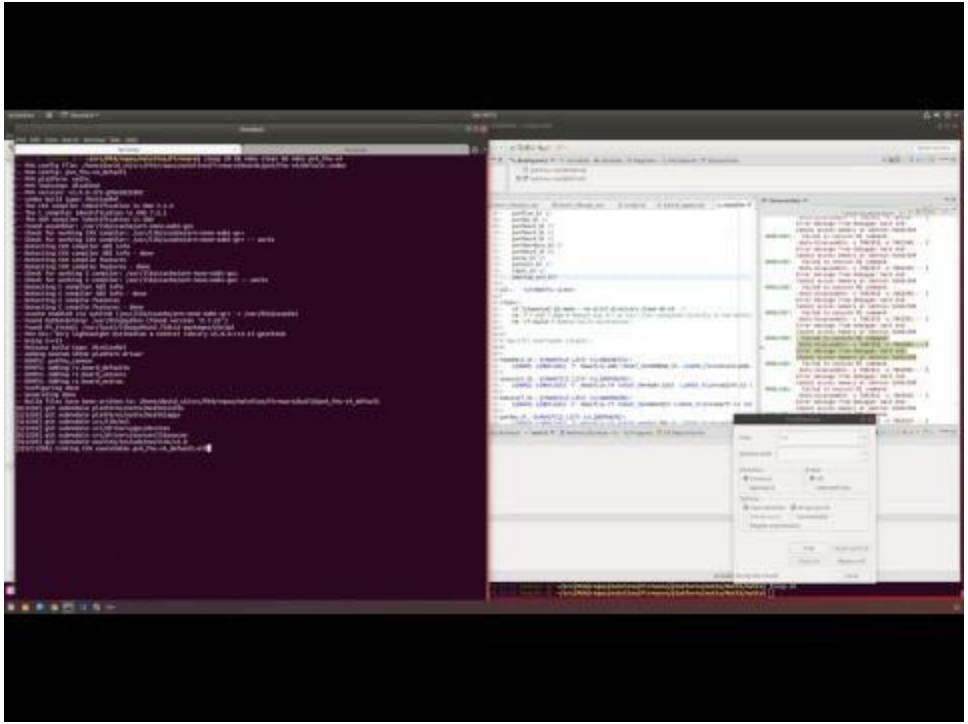
- We build what we can in parallel
- We drive the defconfig to .config process
- We dynamically add to the builtins
- We use the provided magic:
    - CONFIG_ARCH_BOARD_CUSTOM_DIR="../nuttx-config"
    - CONFIG_ARCH_BOARD_CUSTOM_NAME="px4"

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.



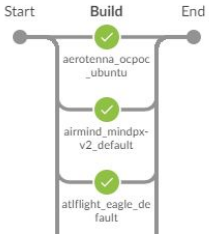
PX4 on NuttX

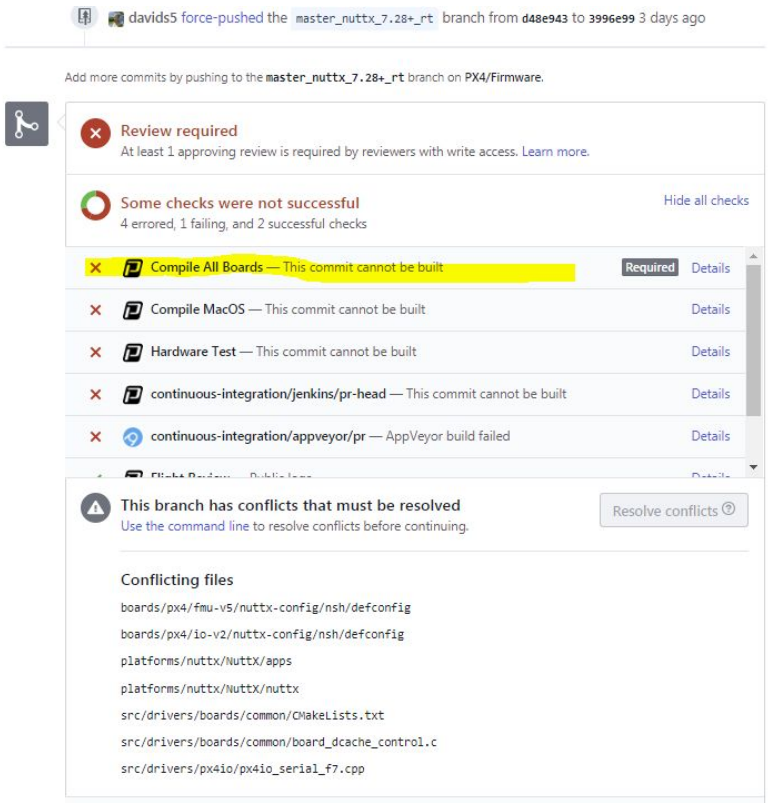


PX4 on NuttX

Test as you code

35 Complete builds in < 11 Minutes





<u>Lean heavily on CI Tools:</u>

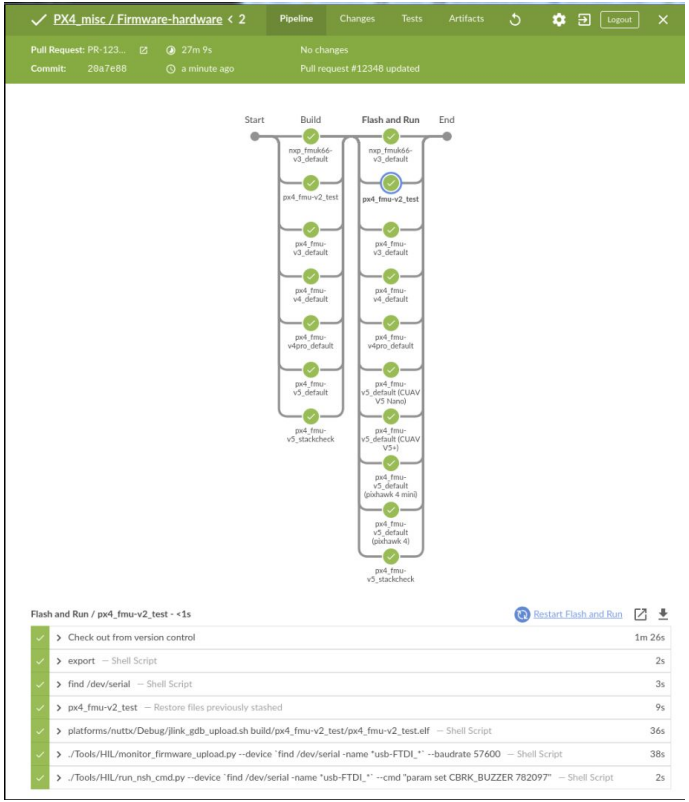
Build all PRs - prevents merging code breaks the build.

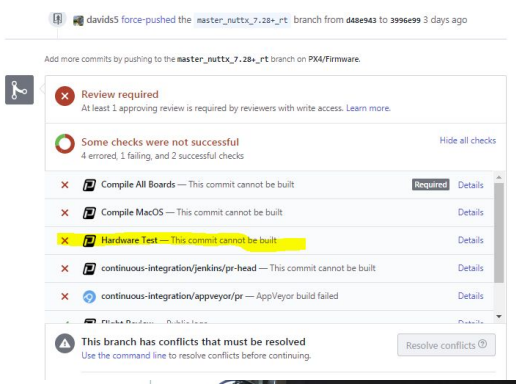
Run a style check - prevents merging code that is not to the coding standard

PX4 on NuttX

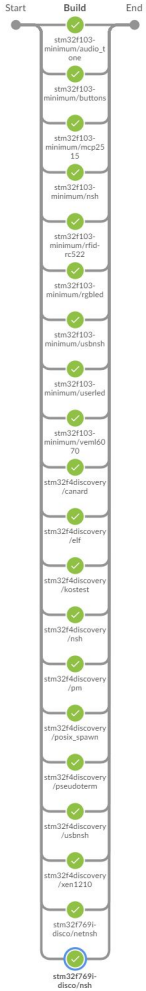
Test as you fly

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

PX4 on NuttX

Some cool PX4 apps

dmesg
hardfault_log
top
uORB

Ideas for future

Fully nested prioritized interrupt structure.
Compile time Device Tree

HardFault Debugging

David Sidrane

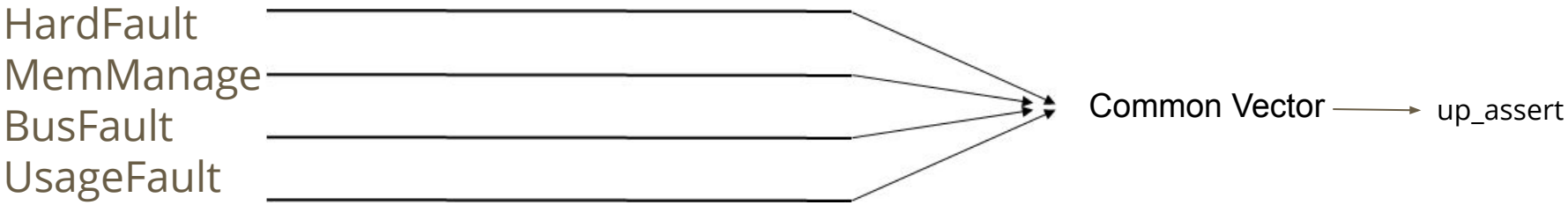
NSCDg

http://www.nscdg.com

Gouda, NL 07/17/2019

What is a HardFault?

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



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

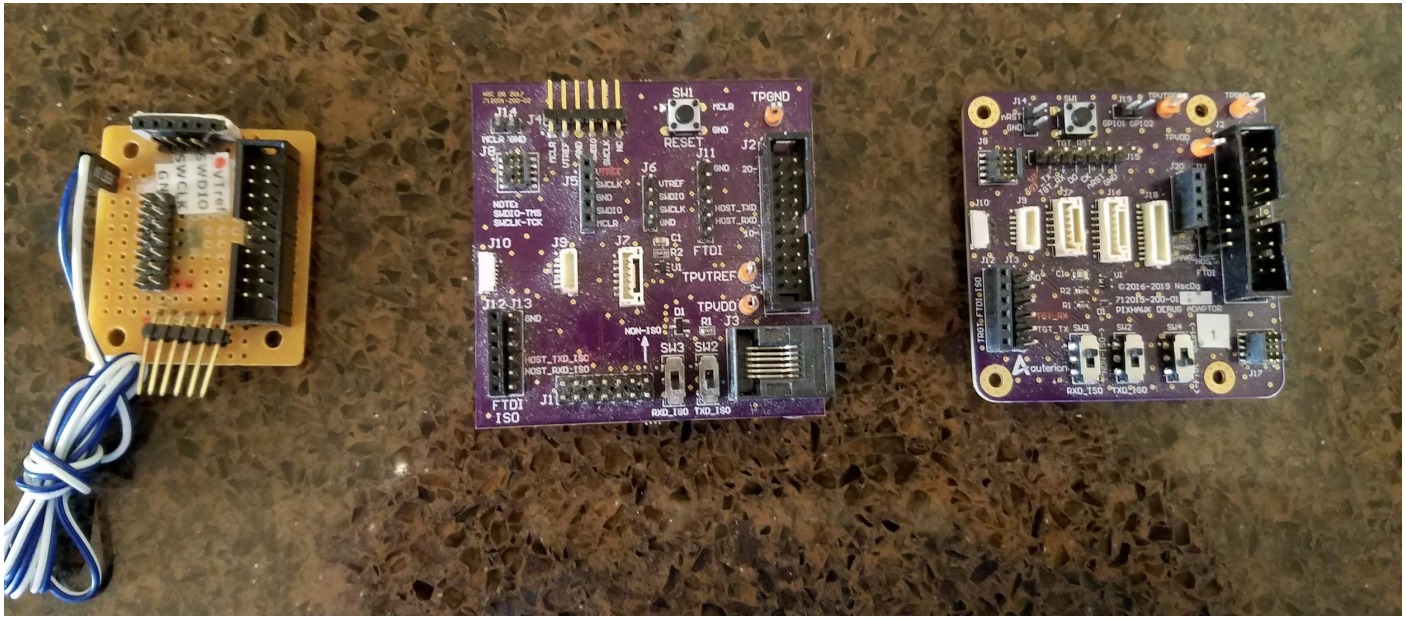
Simple to debug:
(Repeatable occurence of HardFault)

- Hardware accessing a peripheral that is not enabled
- Executing a pure virtual function
- Dereferencing a null pointer

Complex to debug:
(random occurence 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)

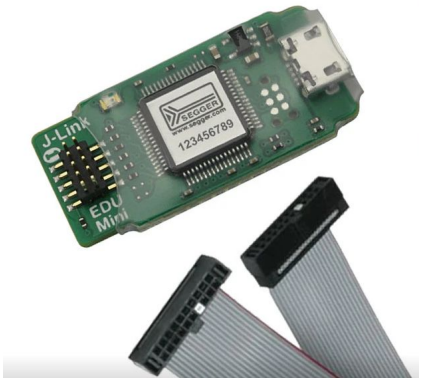
$15,000 USD

=

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