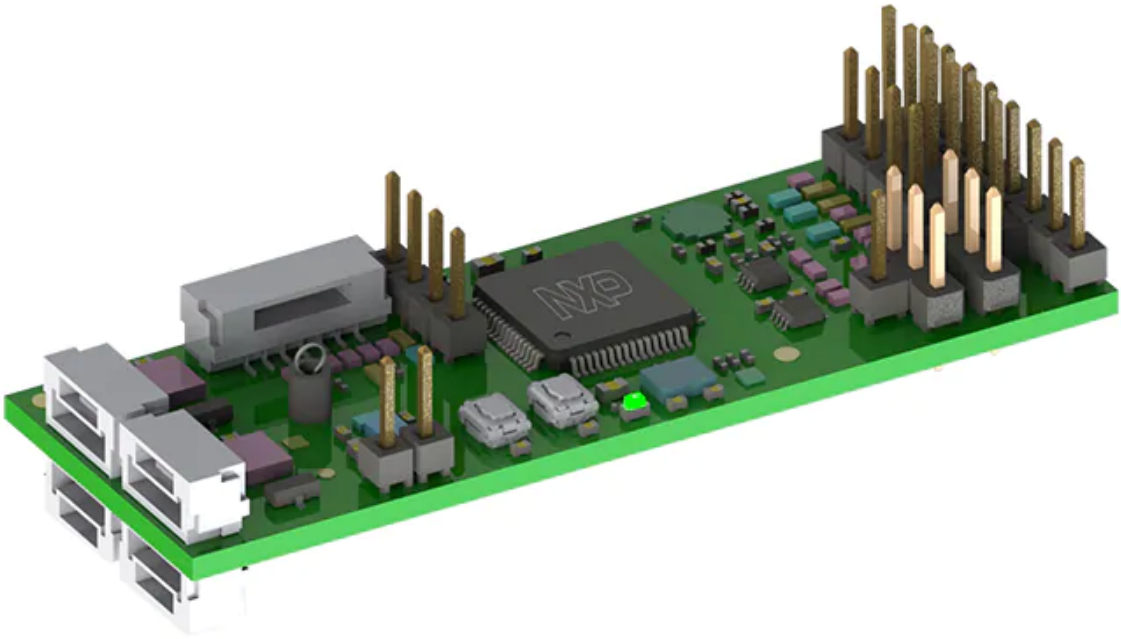
UCANS32K1 Node boards

UCANS32K1 family CAN node boards

This GitBook provides the technical details of the NXP RDDRONE-UCANS32K1 famly of CAN-FD and CAN-SIC boards with UAVCAN protocol support, as well as reference software and examples.

> ⚠ **This GitBook is still a work in progress!**

> See also the NXP Mobile Robotics Community for questions and answers about all our mobile robotics reference designs. There is also thread for questions about UCANS32K148 and UCANS32K1SIC .



> Also have a look at some of the other **NXP GitBooks**:
>
> - HoverGames
> - NavQ Companion Computer
> - RDDRONE-BMS772 Battery Management System
> - D2X Reference Design
> - NXP Cup

What are the UCANS32K1 node boards?

UCANS32K146 and UCANS32K1SIC development boards are general purpose CAN node reference designs. They can be used for any purpose, however specific software has been provided for drones, rovers and other small (autonomous) vehicles. This software allows it to act as a bridge between a CAN bus (with UAVCAN) and I2C, SPI, UART, GPIO or any other pin function of the S32K146 MCU (80 MHz ARM Cortex-M4F, ASIL-B compliant). This allows sensors, actuators and other peripherals to be controlled by other devices on the same CAN bus, such as the RDDRONE-FMUK66 flight management unit reference design.

> The relevant part numbers are:
>
> - KIT-UCANS32K1SIC (complete development kit with **two** UCANS32KSIC boards, a debugger and an adapter board - everything you need to get started!)
> - UCANS32K1SIC (a single UCANS32K146 board, CAN cable and termination network board)
> - KIT-UCANS32K146 (complete development kit with **two** UCANS32K146 boards, a debugger and an adapter board - everything you need to get started!)
> - UCANS32K146-01 (a single UCANS32K146 board, without additional debugger)

Use cases

Possible use cases are:

1. PWM output for motor controllers or servos
   - Relieves the FMU of creating RC-PWM signals
   - Can report information about the motors back to the FMU
2. Battery management systems (also have a look at our BMS772 reference design!)
   - Report power consumption, state of charge, battery health and other faults to the FMU
3. GPS
   - Allows for more than one GPS to be connected to the FMU by communicating GPS info over CAN
4. Sensors
   - Airspeed/pressure sensors can report information to the FMU over CAN
5. And many more
   - Remote lights, arming/safety switches, and really any other peripheral which needs to communicate with the FMU can be connected to the UCANS32K146 development board.

Board specifications

- NXP S32K146 Automotive MCU (80 MHz ARM Cortex-M4F, ASIL-B compliant)
- Dual NXP TJA1044 CAN transceivers
  - OR **Dual NXP TJA1463 CAN-SIC transceivers** (with dual 4-pin JST-GH connectors)
- NXP EdgeLock SE050 secure element with NFC interface (with external antenna, not included)

- One (UCANK1S32K146) or Two (UCANS32KSIC) RC-PWM pin header with optional external power input
- Through-hole solder pads (for 0.100" pin headers) that expose SPI, I2C and UART. Can also be remapped to other pin functions (GPIO, ADC, timer, ...)
- 5V power input; the board can be powered from the 4-pin JST-GH CAN connectors or the 2-pin power input header. There is an optional power input for the RC-PWM header as well specifically for 3 pin connected PWM devices such as high power RC servos.

More information is available on the NXP website.

Hardware designs and example software

The hardware schematics and board layout for UCANS32K146 are available on this GitBook and on the NXP website. We encourage you to create your own designs based on our UCANS32K146 board!

We do not only provide hardware designs, there is also plenty of example software available. There are multiple options to use the UAVCAN protocol. We have also worked with the Apache NuttX and PX4 Autopilot communities to enable their open source software projects on the UCANS32K146.

Additional designs and example software might be made available in the future.

Contribute to this GitBook

We would really like to receive your feedback regarding this GitBook. It is synchronized to a Git repository on GitHub, so you can just open an issue. If you want to contribute you can also open a pull request. The pages are written using an extended version of Markdown, so it should be pretty straightforward to add sections or even complete pages!

Disclaimer

This page contains important information that you should be aware of before using UCANS32K146.

Important Notice

NXP provides the enclosed product(s) under the following conditions:

This reference design is intended for use of ENGINEERING DEVELOPMENT OR EVALUATION PURPOSES ONLY. It is provided as a sample IC pre-soldered to a printed circuit board to make it easier to

access inputs, outputs, and supply terminals. This reference design may be used with any development system or other source of I/O signals by simply connecting it to the host MCU or computer board via off-the-shelf cables. Final device in an application will be heavily dependent on proper printed circuit board layout and heat sinking design as well as attention to supply filtering, transient suppression, and I/O signal quality. The goods provided may not be complete in terms of required design, marketing, and or manufacturing related protective considerations, including product safety measures typically found in the end product incorporating the goods.

Due to the open construction of the product, it is the user's responsibility to take any and all appropriate precautions with regard to electrostatic discharge. In order to minimize risks associated with the customers applications, adequate design and operating safeguards must be provided by the customer to minimize inherent or procedural hazards. For any safety concerns, contact NXP sales and technical support services. Should this reference design not meet the specifications indicated in the kit, it may be returned within 30 days from the date of delivery and will be replaced by a new kit.

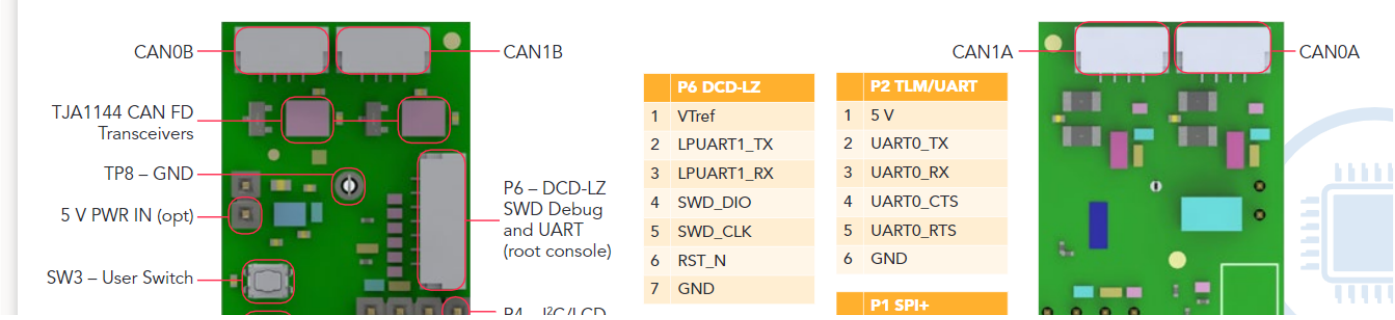
NXP reserves the right to make changes without further notice to any products herein. NXP makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does NXP assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages.

Typical parameters can and do vary in different applications and actual performance may vary over time. All operating parameters, including Typical, must be validated for each customer application by customer’s technical experts.

NXP does not convey any license under its patent rights nor the rights of others. NXP products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the NXP product could create a situation where personal injury or death may occur. Should the Buyer purchase or use NXP products for any such unintended or unauthorized application, the Buyer shall indemnify and hold NXP and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges NXP was negligent regarding the design or manufacture of the part.

PostCard quick reference

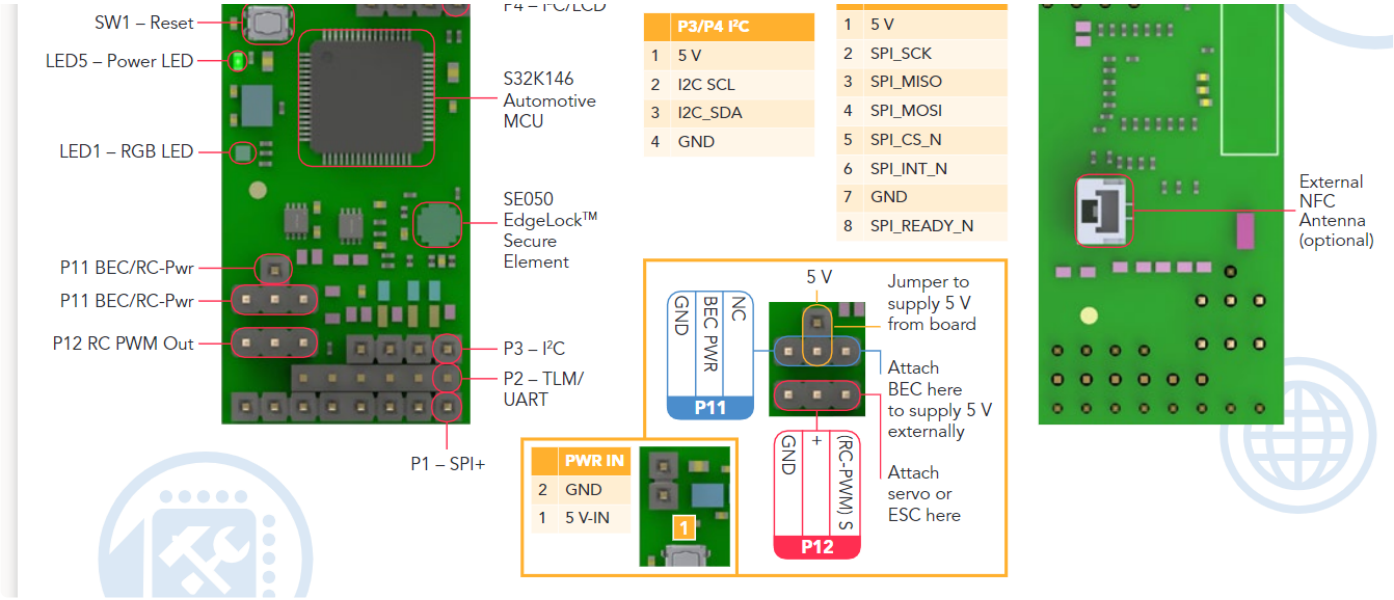
Quick reference to connectors and pinouts for UCANS32K146. CANSIC board follows the same pinout, but includes a second RC-PWM port and connector



	P6 DCD-LZ
1	VTref
2	LPUART1_TX
3	LPUART1_RX
4	SWD_DIO
5	SWD_CLK
6	RST_N
7	GND

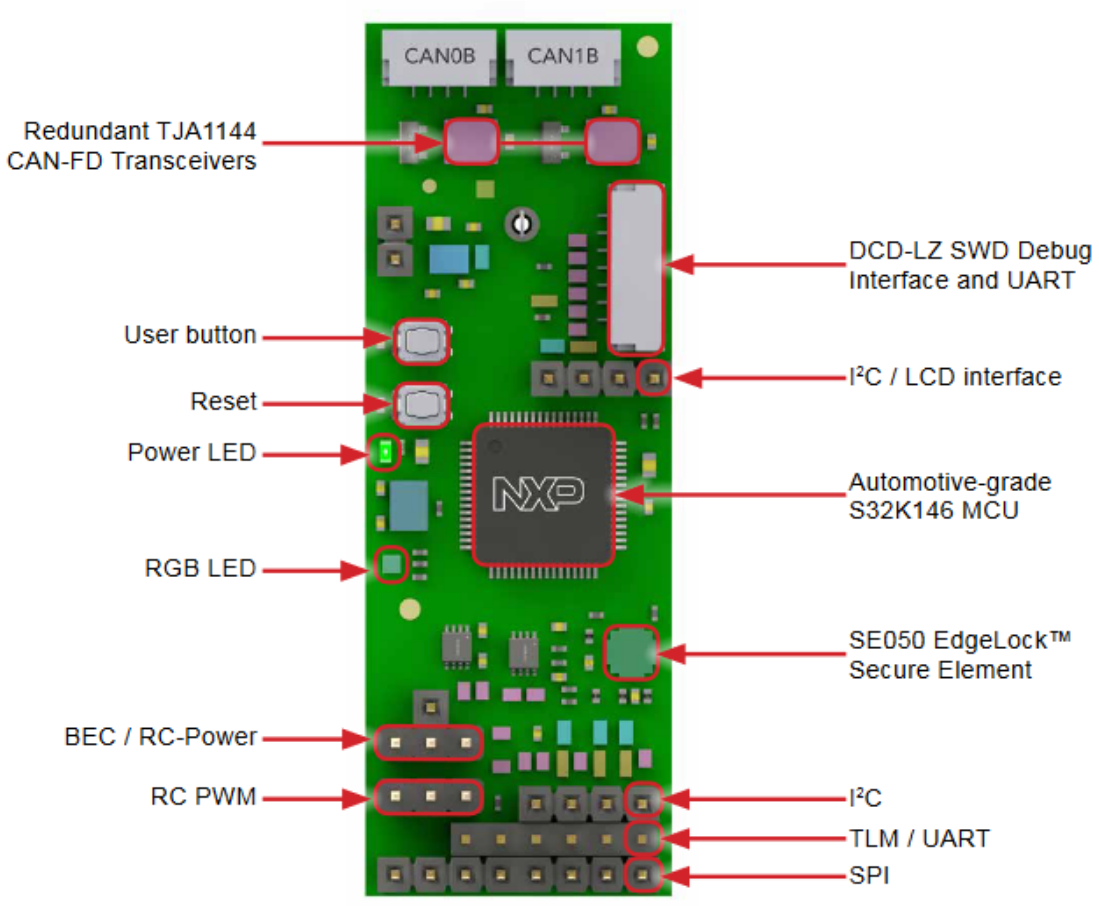
	P2 TLM/UART
1	5 V
2	UART0_TX
3	UART0_RX
4	UART0_CTS
5	UART0_RTS
6	GND

	P1 SPI+

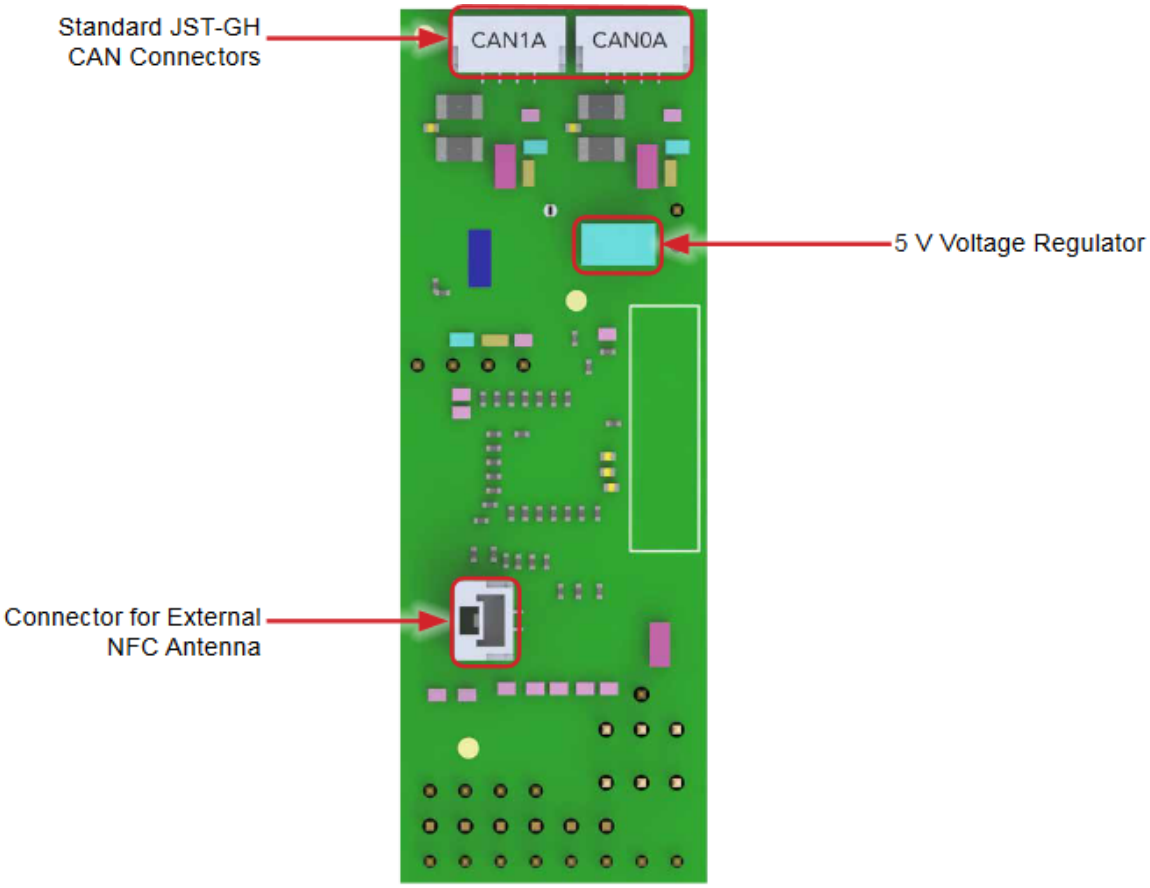


Getting started

UCANS32K146 overview



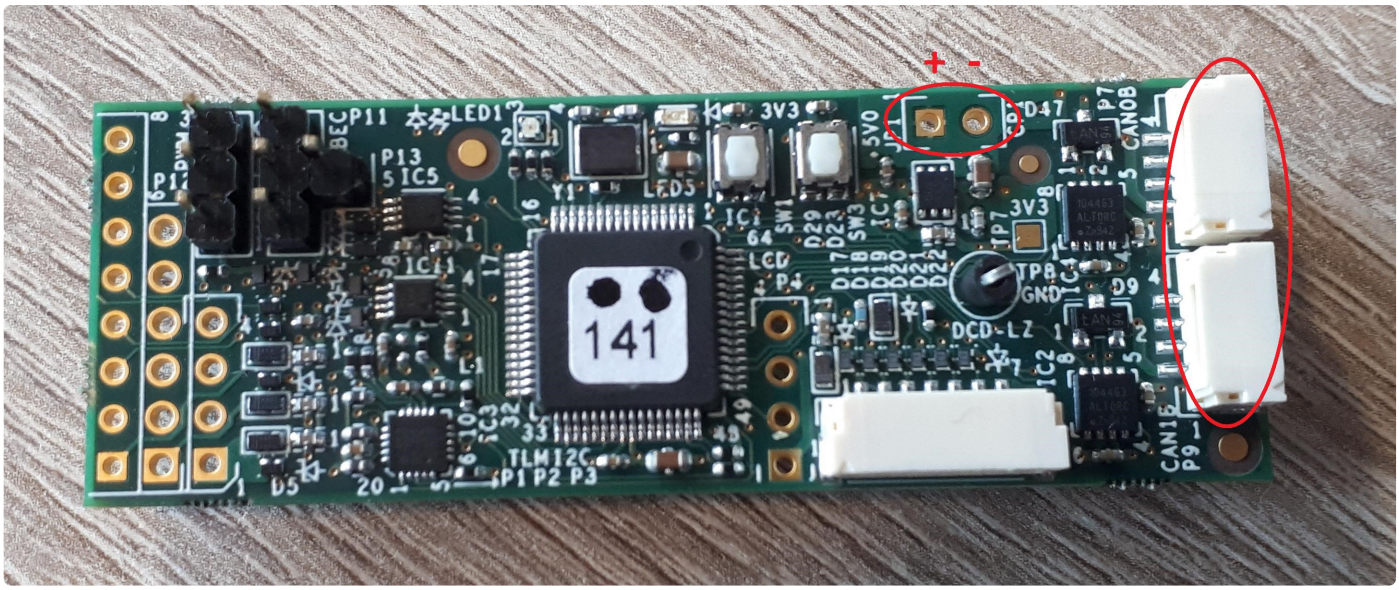
RDDRONE-UCANS32K146 front view



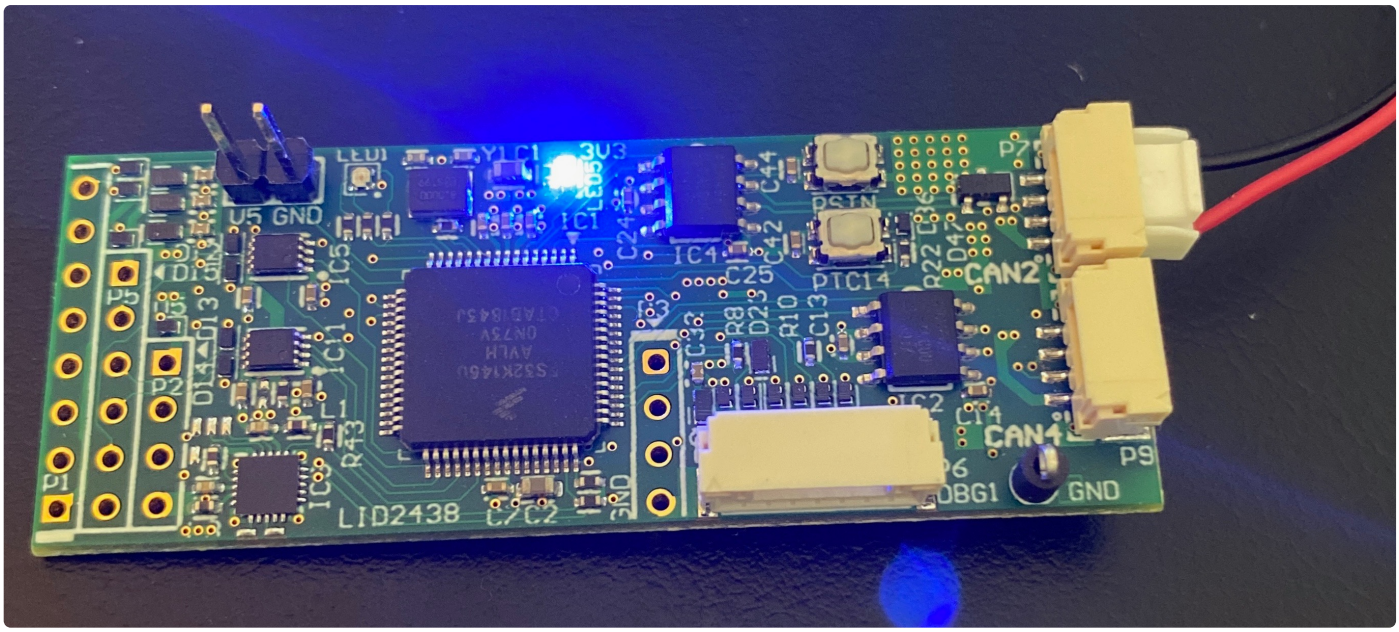
RDDRONE-UCANS32K146 back view

Powering the board

There are two options when powering the UCANS32K146. The first option is to connect 5V power to the power pins on the board. The second option is to power the board through one of the JST-GH CAN connectors. The middle two pins on the CAN connector are for CAN data, and the outer two are for power. The left-most pin is for 5V, and the right-most pin is for GND.



Two options for powering UCANS32K146.

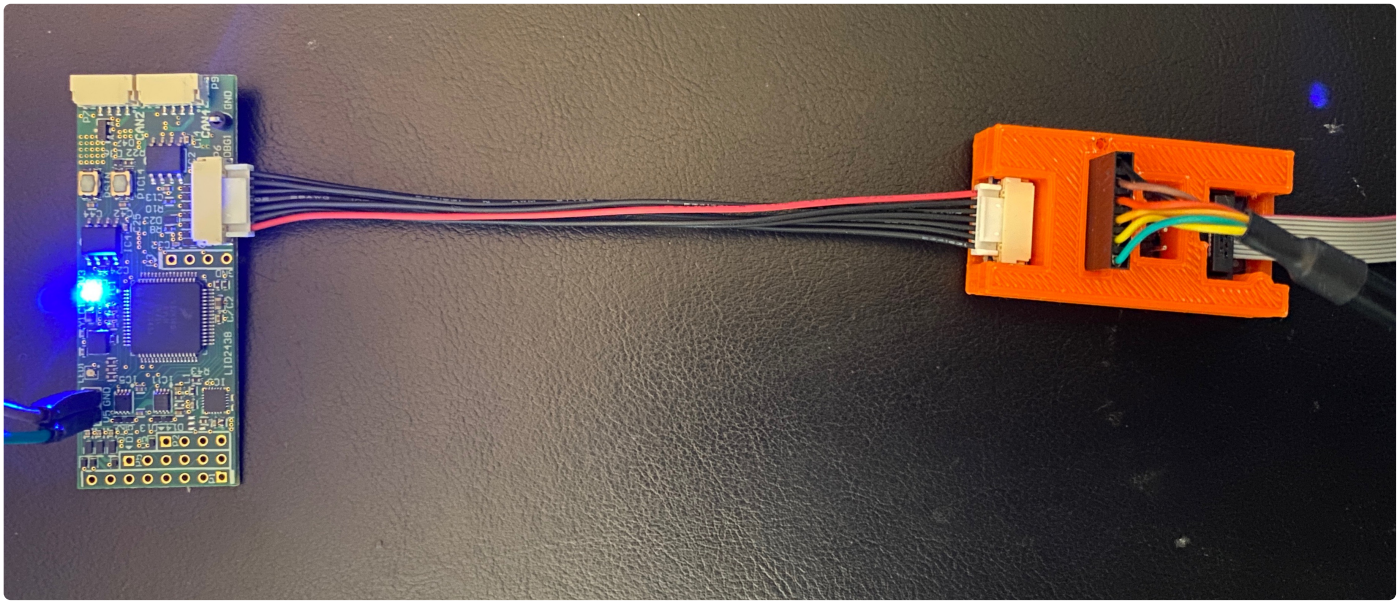


UCANS32K146 (pre-production) powered through the JST-GH CAN connector

---

Flashing and debugging

For flashing firmware and interfacing with the serial console, a 7-pin JST-GH connector with SWD and UART interfaces is present on the board. It is located on the side of the board next to the CAN connectors.



UCANS32K146 (pre-production) connected to a debugger breakout boards.

Guides for flashing binaries to UCANS32K146 are available for PX4 Autopilot and Apache NuttX.

Available software

UCANS32K146 software enablement

PX4 Autopilot

UCANS32K146 is a build target for PX4 Autopilot. PX4 is an open source flight control software for drones and other unmanned vehicles. While UCANS32K146 is not a flight controller, it can leverage the PX4 infrastructure to provide communications and portability of peripheral drivers, leading to enablement of a distributed vehicle architecture. For example drivers for secure element SE050 would be identical on the FMU and CAN nodes. PX4 makes use of a managed and maintained version of NuttX RTOS.

More information

Apache NuttX

In addition to PX4, UCANS32K146 is also a build target in Apache NuttX and can therefore be used without PX4 infrastructure if not needed.

More information

UAVCAN V1

UAVCAN is a lightweight protocol designed for reliable intravehicular communication in aerospace and robotic applications over CAN bus, ethernet, and other robust transports. The name UAVCAN stands for Uncomplicated Application-level Vehicular Communication And Networking. It is created to address the challenge of deterministic on-board data exchange between systems and components of next-generation intelligent vehicles: manned and unmanned aircraft, spacecraft, robots, and cars.

Bare metal example with libuavcan

SocketCAN API

CAN driver compatible with SocketCAN API, takes benefit of POSIX socket API for painless and portable CAN application development.

More information

SLCAN - CAN over serial

This software interface supports debugging UAVCAN and CAN on PC reusing an UCANS32K board reprogrammed as a debugger.

More information

Differences between UCANS32K146-01 and UCANS32K1SIC, UCANS32K146B

Outline the differences in the two versions of the board

Changes as follows:

Type name	CANPHY	PWM	Other
UCANS32K146-01	TJA1044 8 pin (Mantis 2017)	1	
UCANS32K146B	TJA1443 HVSON14 (Avery 2020)	2	
UCANS32K1SIC	TJA1463 HVSON14 (Signal improvement 2020)	2	

UCANS32K1SIC demo application

Demo: UCANS32K1SIC Introduction

A sample application based on PX4/NuttX has been prepared that can be flashed to the UCANS32K1SIC boards. The basic premise is to form a two node CAN-FD network, and give you a console terminal available on each board using

the UART/USB cable connected to your PC.

Simple commands can be issued on each node to initiate CAN-FD communications.

The UCAN boards will be powered via 5V injected through he CAN-TERM board microUSB connection.

DEMO: UCANS32K1SIC Hardware setup

How to connect two UCANS32K1SIC boards in preparation to run the demo software

DRAFT

Show image of PC with USB - UART cable(s) and USB mini power cable connected to CAN-TERM-BRD. Show two UCAN boards connected via the CAN wires. The CAN-TERM boards should be at either end. only one CAN-TERM board should provide 5V power via some USB-Micro cable connection.

NOTE - USB-Micro cable is not included.

DEMO: UCANS321SIC running the demo software

Hardware Reference

Schematics and designs

Schematics of UCANS32K146 prototype and production version

Production Schematics UCANS32K146-01

Production Version 01 Schematics
Schematics for the production version UCANS32K146 are published on NXP.com at the link below under <Documents and Software><Design Resources><Design tools and Files>:

CAN FD development system for Drones, Rovers, and Mobile Robotics
NXP

Prototype Schematics UCANS32K146

Prototype pre-release hardware schematics, board layout and bill-of-materials.

> Note that these files are for the prototype version of the board. These are included here only for convenience to the small number of developers that may have this prototype board.

20191104_SCH_LID2438_final.pdf 549KB
PDF

UCANS32K146 Schematics November 2019

20191105_SCH_BOM_PCB3D.PDF 5MB
PDF

UCANS32K146 SCH, PCB & BOM November 2019

S32K1 SDK

S32K1 SDK with libuavcan

S32 Design Studio and S32K1 SDK

S32 Design Studio IDE for Arm® based MCUs | NXP Semiconductors
NXP

Application Note AN12842

It might take a while for the embedded PDF file to load. The file is also available on NXP.com.

AN12842

Libuavcan S32K1 Driver over CAN-FD

For the UAVCAN Communication Protocol

Rev. 0 — June, 2020 Application Note

by: NXP Semiconductors

Contents

1 Introduction

Libuavcan is a lightweight C++ library for implementing the UAVCAN communication protocol in embedded systems. This application note covers the driver for the transport layer of the protocol over CAN-FD, which utilizes the FlexCAN peripheral available in the S32K1 family of microcontrollers, running at 1 Mbit/s and 4 Mbit/s in nominal and data phases, respectively.

The library is completely statically defined, all the parameters of an application are determined at compile time, avoiding dynamic memory allocation and reducing possible points of failure, and making verification of the system easier.

The last top layers of the protocol implementation for the latest specification, UAVCAN V1.0, are under development at the time this document was written.

The goal of this document is to demonstrate a programming example of the FlexCAN peripheral in particular implementation of the UAVCAN protocol. This for serving as a code reference for custom applications that wish to integrate the CAN-FD capabilities of the module.

2 Overview of UAVCAN

The acronym originally stood as a reference for CAN for Unmanned Aerial Vehicles, but due to the diverse possible applications of the protocol. It later became an acronym for Uncomplicated Application-level Vehicular Communication And Networking. It is an open communication protocol used in avionics, aerospace, robotics and rovers. It is the de-facto protocol in the widely used PX4 autopilot firmware for communications over CAN.

Figure 1. Logo of the communication protocol from which Libuavcan is implemented from

It offers reliable, deterministic and real-time capabilities over the already robust CAN protocol and other transports like UDP, also, no licensing nor approval of any kind is necessary for its implementation. The specification is freely available at the UAVCAN web page, and the source code for many reference implementations are accessible under the MIT license. For links to the specification and code repositories, refer to References.

2.1 Features of the protocol

The protocol's principal abstraction is based on the publisher/subscriber software design pattern. For example, in a robotic system, the sensors would be abstracted as publishers of data and at a predetermined rate, and actuators become subscribers of that information. This pattern is also found in additional robotics software such as ROS.

UAVCAN quick start demo

CAN-FD echo example between two UCANS32K146 boards, sending UAVCAN V1 messages at 1 Mbit/s in nominal phase and 4 Mbit/s in data phase.

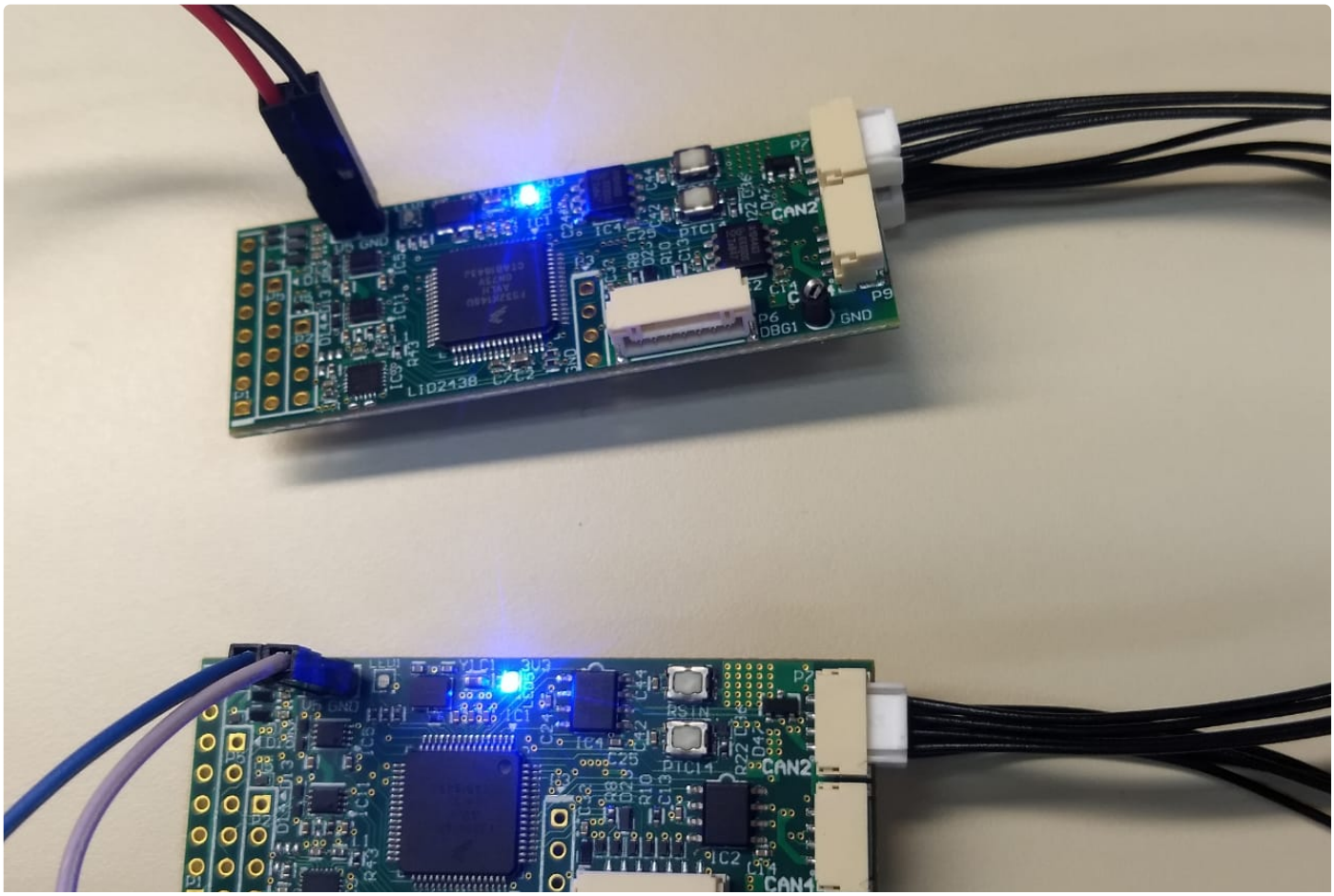
S32 Design Studio for ARM

Download and install S32 Design Studio for ARM. It's available for Windows and Linux.

> S32 Design Studio IDE for Arm® based MCUs | NXP Semiconductors
> NXP

UCANS32K146 hardware setup

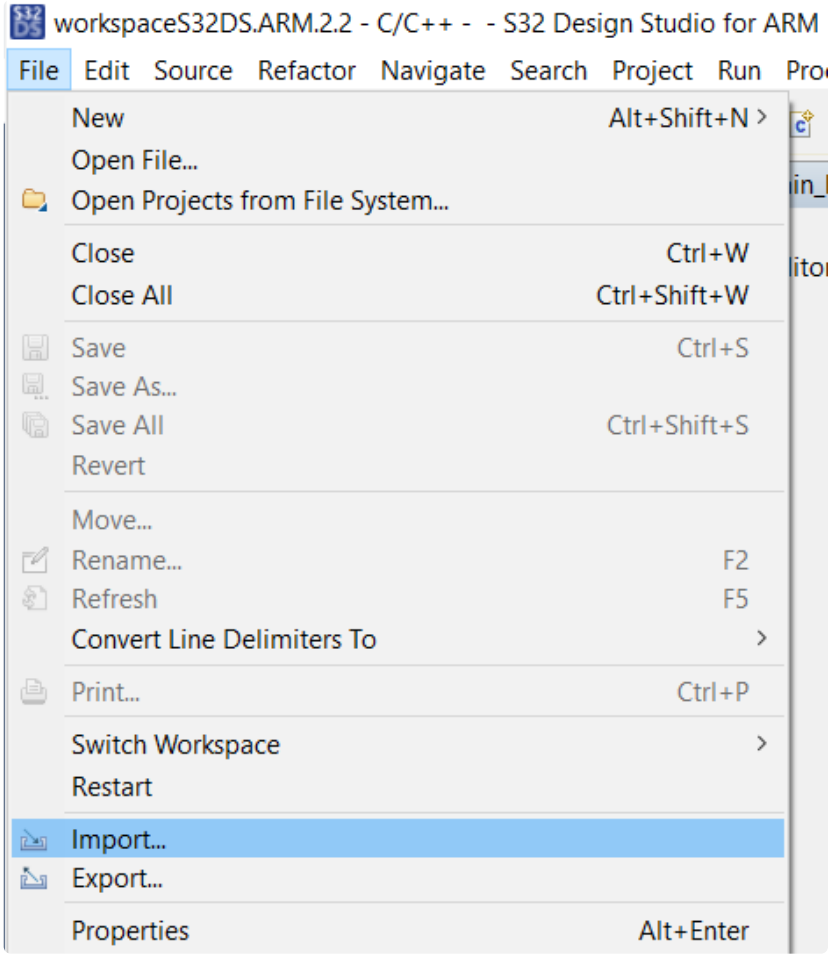
- Interconnect two UCANS32K146 boards with a single 4-wire JST-GH CAN cable between both CAN2 connectors, and CAN bus terminators in the CAN1 connectors below.
- Connect a 5V supply to the pin headers.



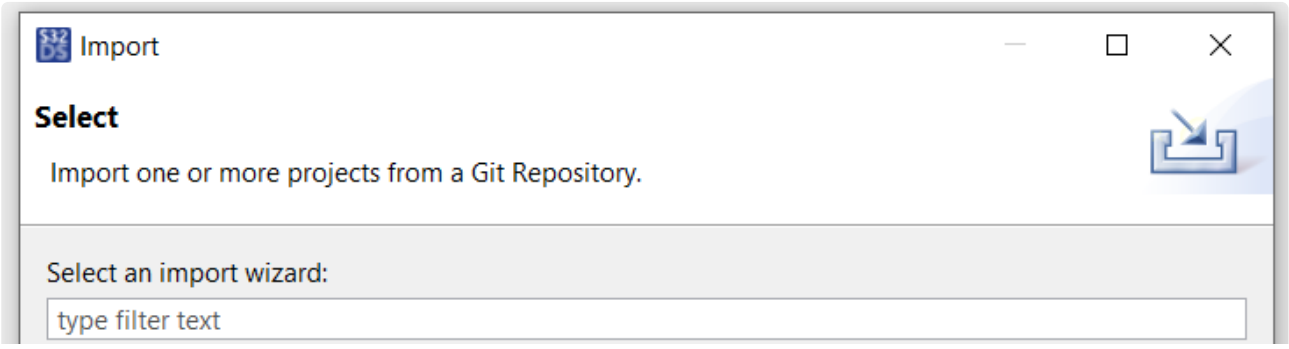
> This pictures was taken with an earlier version of the UCANS32K146 board, there might be some (minor) differences in board layout. The software settings should be the same, though.

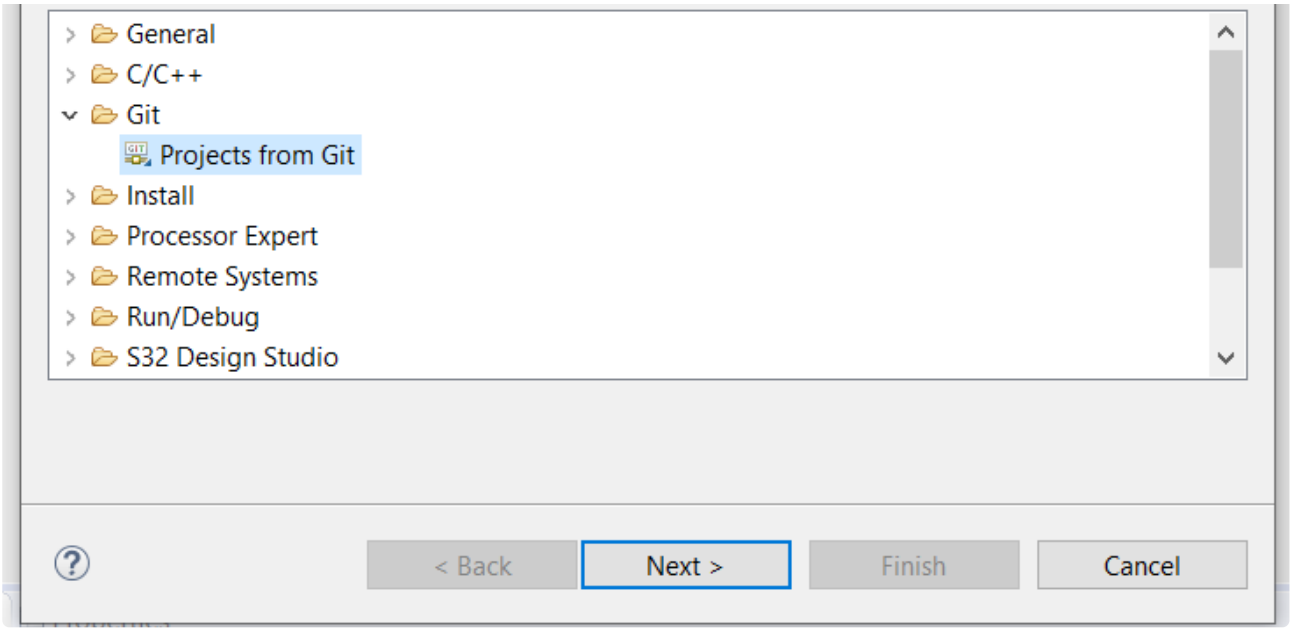
S32 Design Studio example project

1. In S32 Design Studio, go to "File" and then "Import...":

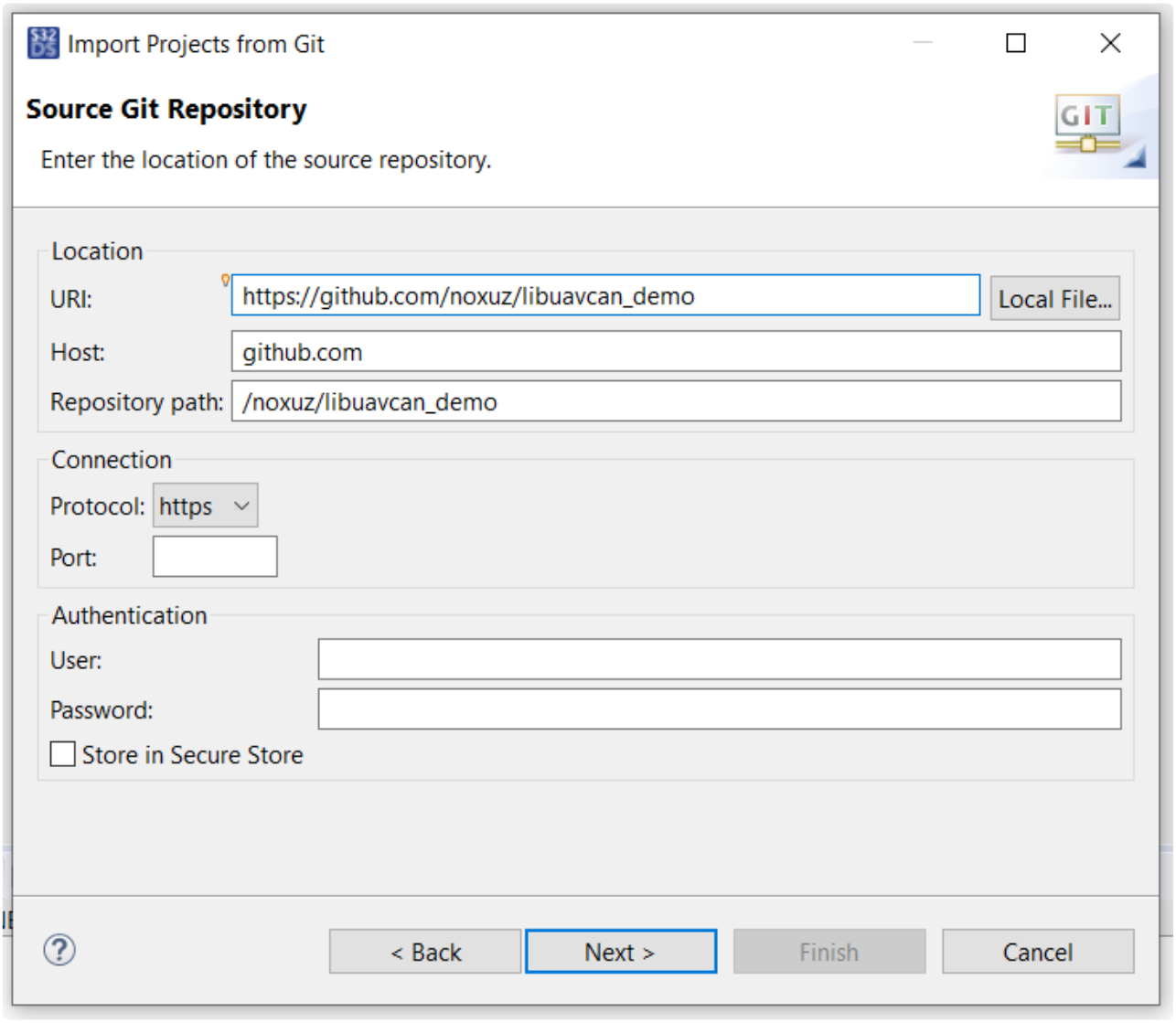


2. Select "Projects from Git" and click "Next":





3. Paste the Git URL (https://github.com/noxuz/libuavcan_demo) of the demo and click "Next":



4. Select the master branch and click "Next".

5. Click "Browse", select the desired destination directory for the project and click "Next".

6. Choose "Import existing Eclipse projects", click "Next" and then "Finish".

7. Click the small arrow at the right of the build icon (the hammer) in the toolbar and select NODE_A to build the code for the first board (transmitter).

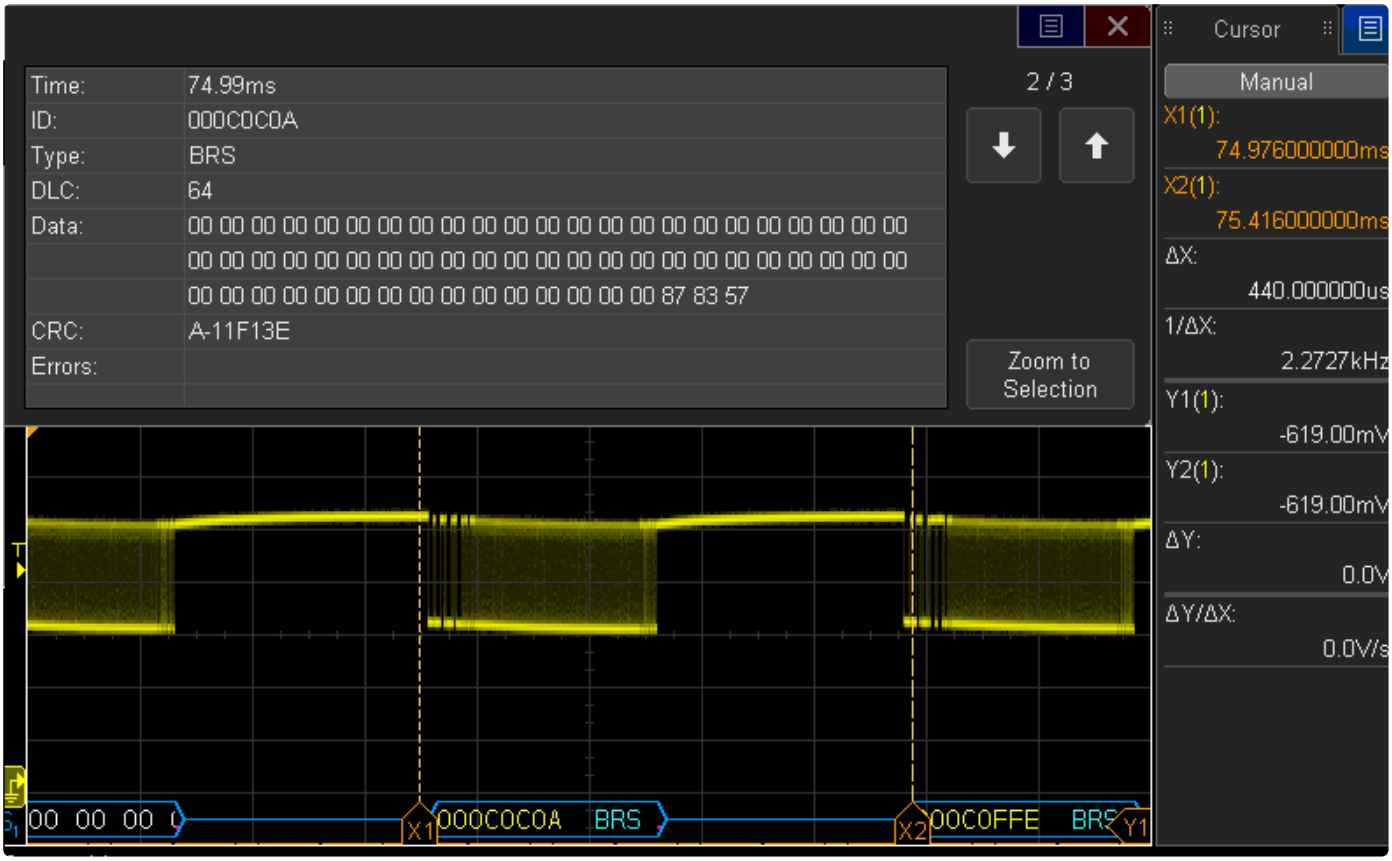
8. Click the yellow lightning shaped icon in the toolbar for flashing the project to the board.

9. In the list located at the left of the popup window, choose the appropriate profile, e.g. "libuavcanV1_demo_Debug NODE_A" if you are building the code for NODE_A. Then click "Flash" with the board connected to the J-Link debugger. Don't forget to power the board (5V)!

10. Repeat steps 7-9 for the other board but with the desired build configuration changed to NODE_B.

11. A green led close to the 5V headers should blink approximately once a second. Also see the description at the top of the `src/main.cpp` file. If the green LED from both boards is not blinking, try pressing the reset button on the board that got the NODE_A program flashed into, which is the one that starts the transmission.

With an oscilloscope or logic analyzer you can view the frames being transmitted at 4Mbit/s data phase speed and at 1Mbit/s in nominal phase:



Apache NuttX

About Apache NuttX

NuttX is an embedded RTOS with an emphasis on standards compliance and small footprint.

What is NuttX?

> NuttX is a real-time operating system (RTOS) with an emphasis on standards compliance and small footprint. Scalable from 8-bit to 32-bit microcontroller environments, the primary governing standards in NuttX are Posix and ANSI standards. Additional standard APIs from Unix and other common RTOS's (such as VxWorks) are adopted for functionality not available under these standards, or for functionality that is not appropriate for deeply-embedded environments (such as fork()). -- ***Apache NuttX website***

NuttX was created by Gregory Nutt and he has been one of the main developers for a very long time.
In 2019 the RTOS was accepted as an incubating project under the Apache Software Foundation, which means it is currently undergoing the process to become an official ASF project. There is now a management committee in place that oversees the ongoing development of NuttX.

NuttX is a very versatile operating system with many (optional) features that supports a wide range of microcontroller platforms. This includes support for NXP's S32K1xx and i.MX RT 10xx, as well as many MCUs from the Kinetis and LPC families.

NuttX is licensed under the permissive Apache License 2.0. This allows it to be integrated into other projects without the need to distribute the derivative work under the same license. Parts of the codebase may still be licensed under the BSD 3-clause license that was used previously, but this should not pose any limitations.

NuttX support for UCANS32K146

A board configuration for RDDRONE-UCANS32K146 is available in the upstream Apache NuttX repositories. As of September 2020, the following features are confirmed to be available:

- Basic support for the NXP S32K1xx family (ARM Cortex-M0+ and M4F)
- SPI, I2C and UART are enabled and can be used in your own drivers and applications
- PWM output using the FlexTimer peripheral is available as well

There is currently no driver support in NuttX for the NXP EdgeLock SE050 secure element.

PX4, NuttX or bare-metal?

PX4 Autopilot is build on top of NuttX, so most of the features that are available in NuttX are also available if you choose to run PX4 on the UCANS32K146. PX4 uses most interfaces that NuttX provides, but adds additional abstractions and in some cases it bypasses the NuttX interfaces and implements its own API. So there's some differences, but you still get most of the features of NuttX, as well as PX4's uORB publish/subscribe messaging API, their parameter system, as well as many drivers for sensors and actuators.

However, if you do not plan on using any of the PX4 features and do not need to interface with any other PX4-enabled system, you can also use the UCANS32K146 development board with "just" NuttX. This makes it easier to customize the configuration and enable features to your liking without having to worry about breaking the PX4 stack.

There is also a third option. NXP offers a "bare-metal" SDK for the S32K1xx family, and there is also a bare-metal libUAVCAN implementation available. That means you can use the UCANS32K146 board with UAVCAN, but without any operating system or software stack. Or you can integrate it with other software.

Documentation and getting help

Documentation for NuttX is available on their website:

- https://nuttx.apache.org/
- https://nuttx.apache.org/docs/latest/
- https://cwiki.apache.org/confluence/display/NUTTX/Documentation

Their online documentation is not exactly beginner friendly. A few volunteers have stepped forward to help improve the available documentation, but there is still some work to be done. Luckily, the code base is pretty clean and self explanatory (and often well documented within the code itself). With a few pointers in the right direction and some patience you can get pretty far on your own.

The community page on the official Apache NuttX website lists a few ways to ask questions, report issues and get in contact with the main developers:

- https://nuttx.apache.org/community/

Download NuttX

How to download (clone) the NuttX source code using Git.

Repositories

NuttX consists of two repositories. The main operating system and an additional set of applications. Both repositories need to be in the same parent folder, with the main operating system being in a folder named "nuttx" and the applications have to be in a folder named "apps".

- **NuttX -** https://github.com/apache/incubator-nuttx
- **Apps -** https://github.com/apache/incubator-nuttx-apps

Let's start by creating a parent folder to store both repositories. The following command will create an "apache-nuttx" folder if it doesn't exist already and makes it the active working directory.

```
mkdir -p ~/src/apache-nuttx && cd ~/src/apache-nuttx
```

We can now clone the Git repository that contains the main operating system. We'll immediately name the folder "nuttx" because the build system might get lost if it has a different name.

```
git clone https://github.com/apache/incubator-nuttx.git nuttx
```

Similarly, we'll clone the apps repository:

```
git clone https://github.com/apache/incubator-nuttx-apps.git apps
```

You should now have a "apache-nuttx" folder located at `~/src/` that contains two folders: "nuttx" and "apps". That's all you need.

Checkout a stable release

By default Git will checkout the latest commit in the master branch when you clone the repositories. This is the latest development version at that point in time. Keep in mind that if you want to keep up with any changes you should use `git pull` to pull in any new commits. Keep in mind to do this for both the "nuttx" and the "apps" repositories.

You can also checkout the latest stable release. Change your working directory to the first repository, and fetch the available tags.

```
git fetch && git fetch --tags
```

Now that the tags are available locally, you can checkout a particular release. At the time of writing the latest stable release is NuttX 9.1.0. You can checkout this particular version with the following command:

```
git checkout nuttx-9.1.0
```

Don't forget to do both steps for both repositories! A particular version of the NuttX OS always should be matched by the same version of the NuttX Apps.

Building and flashing NuttX

Toolchain

The online NuttX documentation provides an overview of possible development environments, but the README file in the main NuttX repository is also a good starting point. In general you're good to go if you are using Linux with a standard GNU toolchain.

Within the NXP Mobile Robotics team we often use PX4 Autopilot as well. They have their own toolchain preferences (which are compatible with NuttX), so we often install their preferred toolchain. This is also easy to do because they provide some scripts that take care of most of the work.

Building NuttX

Building NuttX is very easy when you have all required tools installed. You just have to select a canned configuration, change the configuration to your liking and just run `make`. If you want to throw away the current configuration and select another configuration you first have to run `make distclean`. That's all.

Flashing NuttX

Download and install the J-Link Software and Documentation Pack. It is available for all major operating systems, but we will assume here that you are using a Linux-based OS. Open a terminal and navigate to the main `nuttx` directory, which contains `nuttx.bin` after a successful build. Then you can start the J-Link tools by entering:

```
1 JLinkExe
```

Make sure that the debugger is plugged into both the UCANS32K146 board and your computer. If you are using a virtual machine, the debugger USB device should be made available inside the VM. Also do not forget to power the board. Now enter:

```
1 connect
```

You are now asked to specify a device. It is quickest to manually enter the device:

```
1 s32k146
```

The target interface needs to be specified, which is SWD:

```
1 s
```

Finally you have to specify the target interface speed. It is recommended to use 1000 kHz:

```
1 1000
```

Now flash the binary with:

```
1 loadbin nuttx.bin 0
```

The binary will be programmed and this process will also be verified. It should then mention if everything went OK. You can quit the J-Link tools with:

```
1 q
```

You may need to power cycle the device afterwards. The debug console should now be available on LPUART1 (115200 baud) - which is also accessible on the debugger breakout board.

NuttX configuration

NuttX offers a wide range of configuration options and features that can be enabled with one of the available Kconfig frontends.

Install Kconfig frontends

NuttX uses a similar configuration system as the Linux kernel. There are Kconfig files that define the different options and you can generate a configuration file that can be used by the build system with a Kconfig frontend.

NuttX provide their own fork/back-up of the Kconfig-frontends, to make sure that they always have access to a compatible version. It is still hosted on their old BitBucket repository.

> In Ubuntu 20.04 it is possible to directly install the kconfig-frontends package from the package repositories. You only need to run `apt install kconfig-frontends`.
>
> You can then skip most of the instructions below.

First, make sure you have gperf installed:

```
1 sudo apt install gperf
```

Then download the source code by cloning the NuttX Tools repository from BitBucket:

```
1 git clone https://bitbucket.org/nuttx/tools.git
```

Change the working directory to the kconfig-frontends folder:

```
1 cd tools/kconfig-frontends
```

Configure the build to include (at least) the menuconfig and qconfig tools. The Kconfig frontends will be installed into the /usr folder.

```
1
```

```
./configure --enable-mconf --enable-qconf --prefix=/usr
```

Now build the tools according to the configuration that we just created:

```
make
```

And finally install the Kconfig-frontends that we just build from source:

```
sudo make install
```

Once you have a "default" configuration in place (we will get to that in a second), you can edit the configuration with `make menuconfig` or `make qconfig`. Both tools have their pros and cons, just give them a try and see which one you like the most!

Start with a canned configuration

Make sure you are inside the "nuttx" folder:

```
cd ~/src/apache-nuttx/nuttx
```

Then use the configure script to select the "nshdebug" configuration for the "rddrone-uavcan146" board:

```
./tools/configure.sh rddrone-uavcan146:nshdebug
```

Make sure to first run `make distclean` if you still have another active configuration.

Configuration

Menuconfig is the "default" tool to configure the NuttX build. Once you have a board configuration selected (see above) you can edit the configuration with:

```
make menuconfig
```

You can also use qconfig, which shows the configuration as a nested list instead of different menus. This might make it a bit easier to navigate through the many options that are available:

```
make qconfig
```

Menuconfig and qconfig are easy to use tools, but NuttX is very configurable and you can easily got lost in the hundreds of menus and options that it provides. Take your time to explore the options that NuttX has to

offer, but don't try to enable many options at once. Just select a few, build the code and give it a try on the

SocketCAN

What is SocketCAN?

> SocketCAN is a set of open source CAN drivers and a networking stack contributed by Volkswagen Research to the Linux kernel. Formely known as Low Level CAN Framework (LLCF). -- ***Wikipedia***

⚠ This page is still under construction.

SLCAN

What is SLCAN?

SLCAN is an interface for CAN communication over a serial line.

ⓘ **November 2020:** SLCAN support is not yet publicly available. We hope to release this as soon as possible, but it will likely take us at least a few more weeks.

⚠ This page is still under construction.

PX4 Autopilot

About PX4 Autopilot

PX4 is an open source flight control software for drones and other unmanned vehicles.

What is PX4 Autopilot?

> PX4 is an open source flight control software for drones and other unmanned vehicles. The project provides a flexible set of tools for drone developers to share technologies to create tailored solutions for

> drone applications. PX4 provides a standard to deliver drone hardware support and software stack, PX4 is part of Dronecode, a non-profit organization administered by the Linux Foundation to foster the use of open source software on flying vehicles. Dronecode also hosts QGroundControl, MAVLink & the SDK. -- ***PX4 website***

> ✓ PX4 Autopilot for UCANS32K146 is build on top of NuttX. Therefore, most of the information in the "Apache NuttX" section also applies to PX4! Besides flight control and autopilot software, it also contains additional tools, drivers and middleware.

Why use PX4 Autopilot on UCANS32K146?

> The UCAN board is not an Autopilot, so why consider using PX4 on it? There are a number of reasons for this:
>
> - Building a PX4 distributed architecture. The same peripheral drivers running on an FMU can be reused here on the UCAN board, with only the CAN bus (UAVCAN) separating them. This means a common codebase is developed and used for something like a sensor or actuator.
> - PX4 carefully maintains and updates their branch of NuttX, and regularly backports to mainstream NuttX. This means PX4/Nuttx is more stable and there are minimal "surprises" due to untested code making it's way into the OS.
> - PX4 includes and tests additional tools and methods within their distribution. i.e 'top'
> - Connection with the PX4.io community ecosystem through slack, discourse and regular standards bodies meetings.

PX4 support for UCANS32K146

The PX4 development team provides basic support for UCANS32K146. A board configuration is available from which a binary can easily be build. Because PX4 is build on top of NuttX, it supports most of the features that were enabled in the NuttX build for UCANS32K146.

There are some additional features offered by PX4:

- uORB and MAVLink messaging
- PWM generation
- Most PX4 drivers and modules can be enabled

There is currently no driver support in PX4 or NuttX for the NXP EdgeLock SE050 secure element.

Documentation and getting help

Documentation for PX4 is available on their website:

- https://px4.io/
- https://docs.px4.io/master/en/
- https://dev.px4.io/master/en/

PX4 Autopilot is originally a drone flight control stack, therefore most of the documentation is focused on using PX4 on flight management units. Not all features may be directly available on UCANS32K146, though you can potentially enable all modules in the source code.

Issues can be reported on the PX4 GitHub. There are various support channels available where you can ask questions and discuss your ideas.

Building and flashing PX4

How to build and flash PX4 Autopilot for NXP UCANS32K146.

Prerequisites

You need a build environment that can build PX4 Autopilot. You can generally use the same environment as with Apache NuttX, but you will need to install some additional packages. To get started, you can use the HoverGames virtual machine, or install your own Linux (virtual) machine. When you have a basic Linux environment, you only need to install the PX4 toolchain to continue.

More information is also available in the PX4 Developer Guide.

If you have not yet cloned the PX4 source code as part of the toolchain installation, you should do so now:

```
1 git clone https://github.com/PX4/PX4-Autopilot
```

Building PX4 Autopilot for UCANS32K146

Change your working directory (`cd` command) to the PX4-Autopilot Git repository that you just cloned. Next we will build the bootloader and firmware for our UCAN board.

Bootloader

> ⓘ You only need to build and flash the bootloader once. If you rebuild/reflash PX4, you can skip all of the bootloader steps.

Downloading the Bootloader

If you don't want to build the bootloader, you can download it by clicking the file below.

http://ci.px4.io/job/PX4_misc/job/Firmware-compile/job/master/lastSuccessfulBuild/artifact/buil…

UCANS32K146 PX4 Bootloader (Latest build)

Alternatively, run the following command to build the bootloader:

```
1 make nxp_ucans32k146_canbootloader
```

The binary file will be located at `PX4-Autopilot/build/nxp_ucans32k146_canbootloader/nxp_ucans32k146.bin`. Keep this file handy for flashing later.

PX4 Firmware

A prebuilt version of PX4 for UCANS32K146 is linked below. This is the latest build of PX4 master.

http://ci.px4.io/job/PX4_misc/job/Firmware-compile/job/master/lastSuccessfulBuild/artifact/buil…

UAVCAN PX4 firmware binary (Latest build)

Alternatively, you can build the firmware by running the following command in the root of the PX4-Autopilot repository:

```
1 make nxp_ucans32k146
```

The .bin file for flashing the firmware will be stored in `build/nxp_ucans32k146_default/deploy`. The file you're looking for is `34.bin`. You can leave the file there or copy it to another location for flashing the board in the next section.

Flashing PX4 Autopilot to the UCANS32K146 board

Download and install the J-Link Software and Documentation Pack. It is available for all major operating systems, but we will assume here that you are using a Linux-based OS. Open a terminal and navigate to the directory which holds the `nxp_ucans32k146.bin` file that we build in the previous step.

Now start the J-Link tools by entering:

```
JLinkExe
```

Make sure that the debugger is plugged into both the UCANS32K146 board and your computer. If you are using a virtual machine, the debugger USB device should be made available inside the VM. Also do not forget to power the board. Now enter:

```
connect
```

You are now asked to specify a device. It is quickest to manually enter the device:

```
s32k146
```

The target interface needs to be specified, which is SWD:

```
s
```

Finally you have to specify the target interface speed. It is recommended to use 1000 kHz:

```
1000
```

Now flash the bootloader with:

```
loadbin /path/to/nxp_ucans32k146.bin 0x0
```

The binary will be programmed and this process will also be verified. It should then mention if everything went OK.

Next, flash the PX4 firmware binary with:

```
# If using pre-built binary
loadbin /path/to/34-0.1.{commit}.uavcan.bin 0x6000
# If using self-built binary
loadbin /path/to/34.bin 0x6000
```

And you're good to go.

You can quit the J-Link tools with:

```
q
```

You may need to power cycle the device afterwards. The debug console should now be available on LPUART1 (115200 baud) - which is also accessible on the debugger breakout board.

More information

The PX4-build for UCANS32K146 is in an early state and not all features may be available. More information about PX4 Autopilot is available in their User Guide and Developer Guide. Their software development is managed on GitHub. There are also various support channels available.

PX4 examples to try

Assuming you have successfully connected via the UART and are seeing the PX4/NuttX nsh> prompt, you can try a few things using PX4/NuttX

```
nsh> help
```

This will show you basic help as well as list the Builtin Apps.

Try running some of the builtin apps: (Lines beginning with # below are just comments)

```
nsh> #run hello world
nsh> hello

nsh> #control the onboard RGB LED
nsh> led_control breathe -c cyan

nsh> #check the actual RGB PWM driver status
nsh> rgbled_pwm status

nsh> #view the rtos processes
nsh> top

nsh> #check the hardware version and git verion
nsh> ver hw
nsh> ver git
```

As you can see there are also a number of other builtin apps ready to test servo's, I2C devices, SPI devices, and even a GPS when attached.