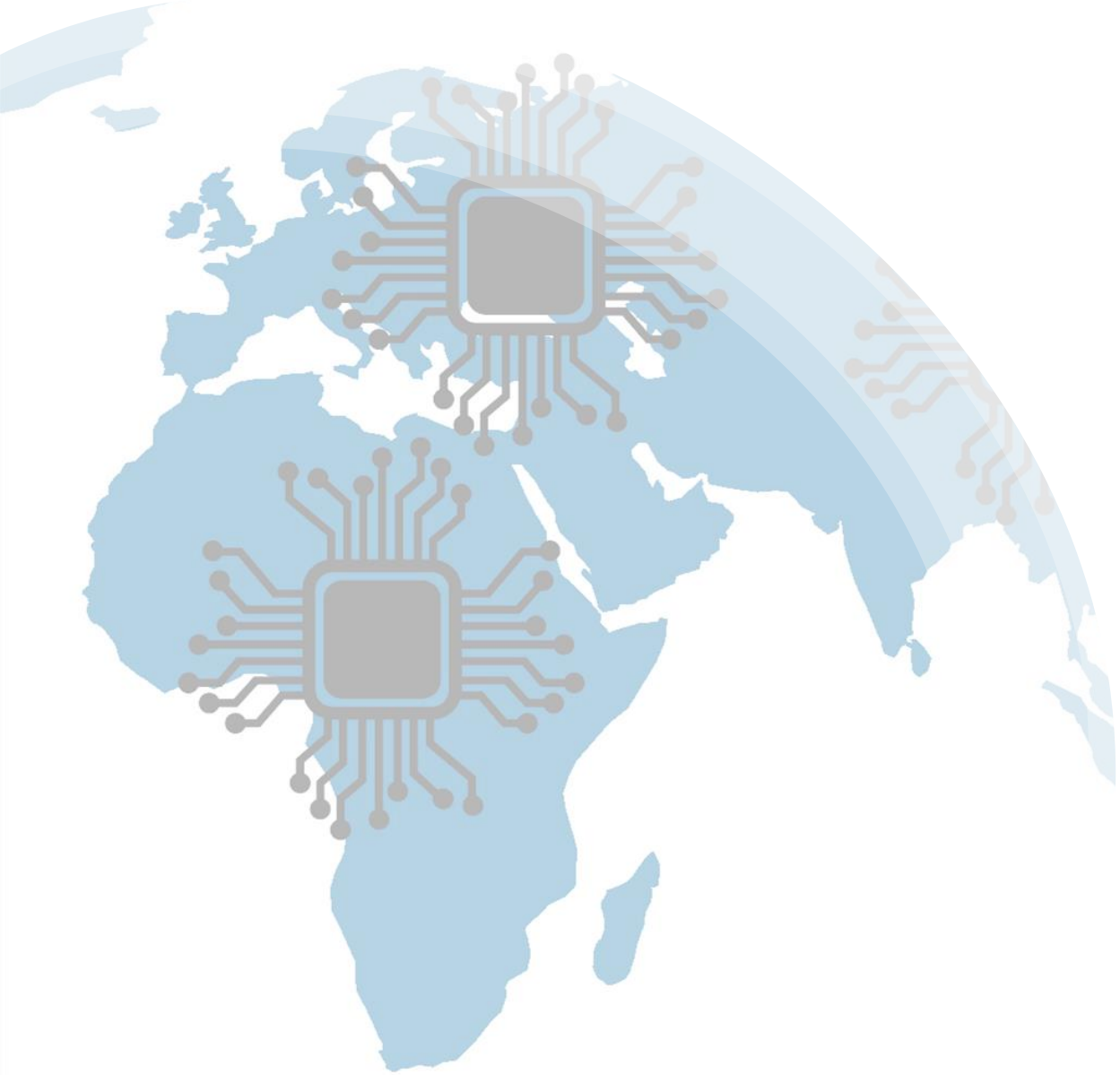
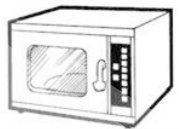


SocketCAN in NuttX

Peter van der Perk

August 15-16 2020

NuttX Online Workshop



Mobile robotics at NXP

Small Drone team -> now Mobile robotics.

- Drones
- Rovers
- Delivery vehicles and open experimentation like X-VTOL

Applying products from throughout the company to NuttX and PX4. MCU, MPU, Safety, Security, Networking, Wireless.

Automotive functional safety parts:

- ISO26262, ASIL diagnostics and fail operational as well as semiconductor build rules.

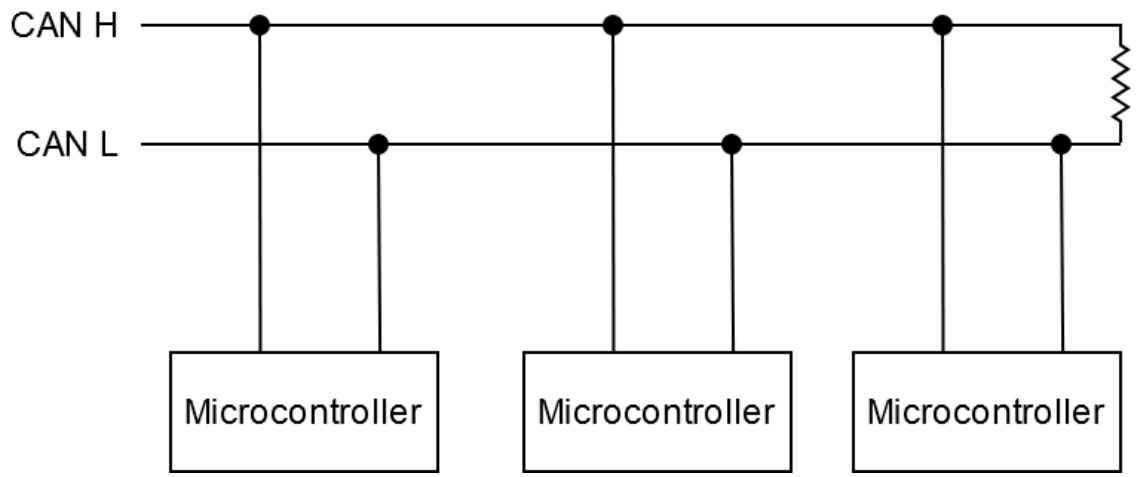
“We don’t build drones we build reference designs”

Outline

- What is CAN bus?
- Existing CAN subsystem in NuttX
- SocketCAN in NuttX
- SocketCAN Demo
- Where do I use NuttX/SocketCAN for?
- Conclusions

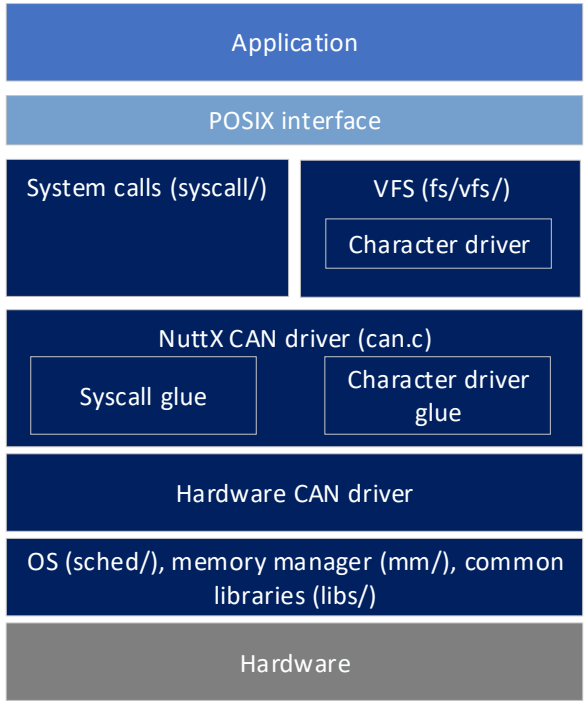
What is CAN bus?

- CAN stands for **Controller Area Network**
- It's a bus topology, single pair of wire for multiple controllers
- Communication goes through so-called CAN-frames which contains an identifiers which also acts a priority and a payload.



Existing CAN subsystem in NuttX

- Character device based driver
- ioctl() interface, for device configuration
- NuttX specific API (Not a standardized API)

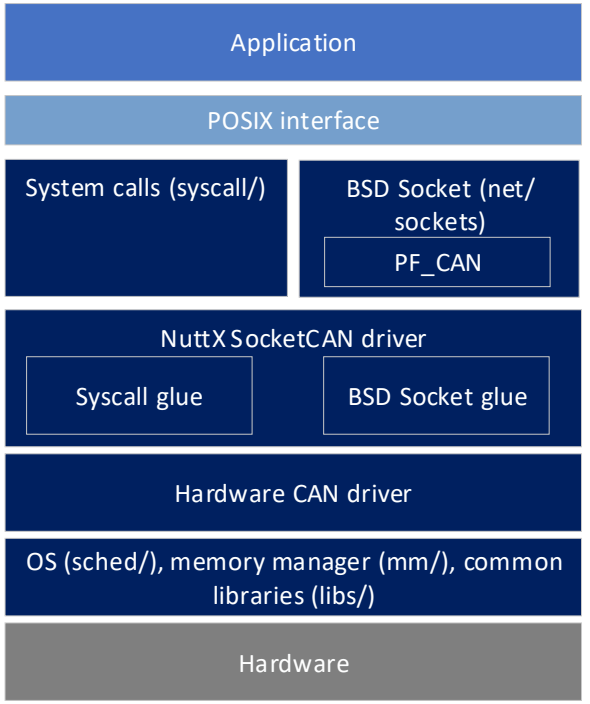


NuttX CAN

SocketCAN proposal

- Use of the SocketCAN API created by Volkswagen Research
- SocketCAN provides a multiuser capable as well as hardware independent socket-based API for CAN based communication and configuration.

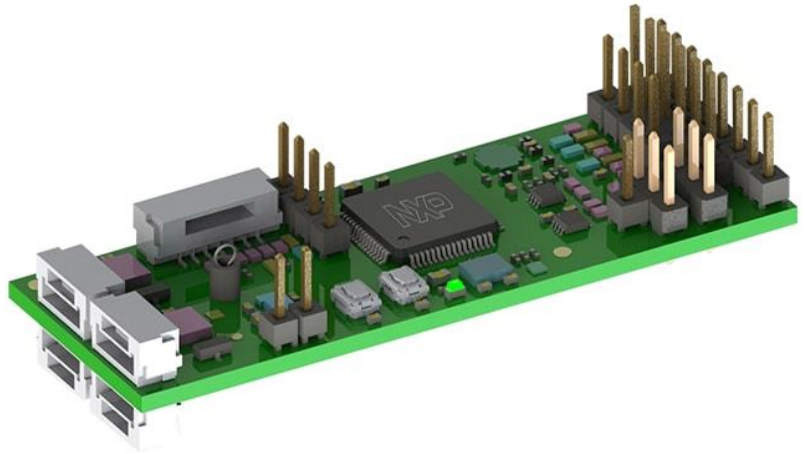
- NuttX already supports the socket interface
- Extend Socket API with PF_CAN family
- CAN stacks/application can be more easily integrated in NuttX
  - Canutils
  - UAVCAN
  - CANopen
  - Any POSIX compatible SocketCAN application

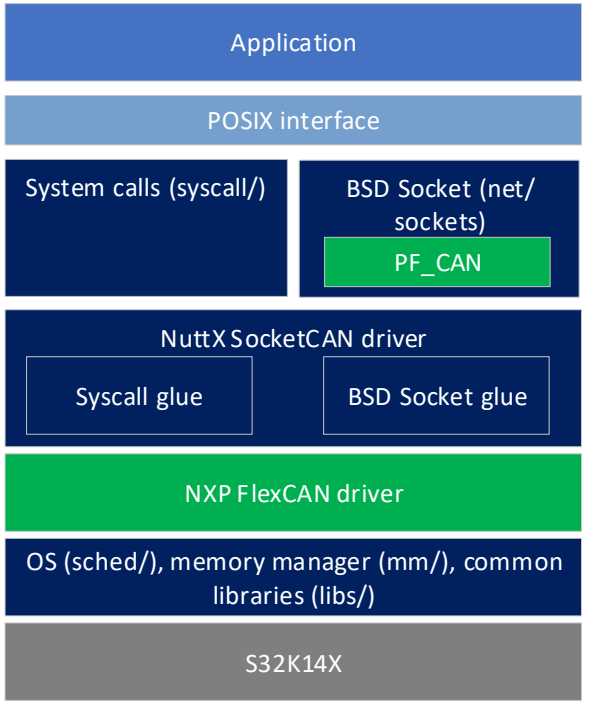


NuttX SocketCAN

SocketCAN test vehicle

- RDDRONE-UCANS32K146
  - S32K146 MCU, already runs NuttX
  - 2x CAN controller with CAN2.0B and CAN FD support
- Tasks:
  - Extend NuttX Network stack with the PF_CAN protocol stack
  - Write S32K1XX FlexCAN driver
  - Port some demo apps



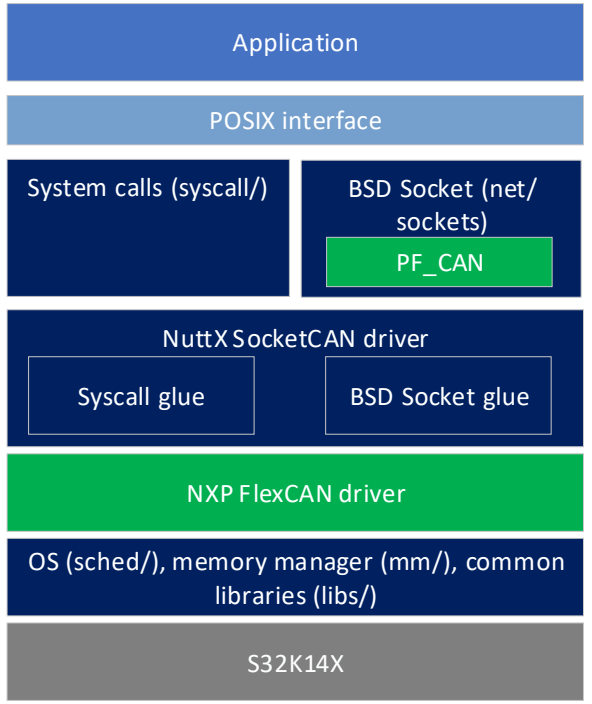


NuttX SocketCAN

Adding the PF_CAN protocol stack

Creating net/can (Thank you Gregory Nutt for creating the initial stubs)

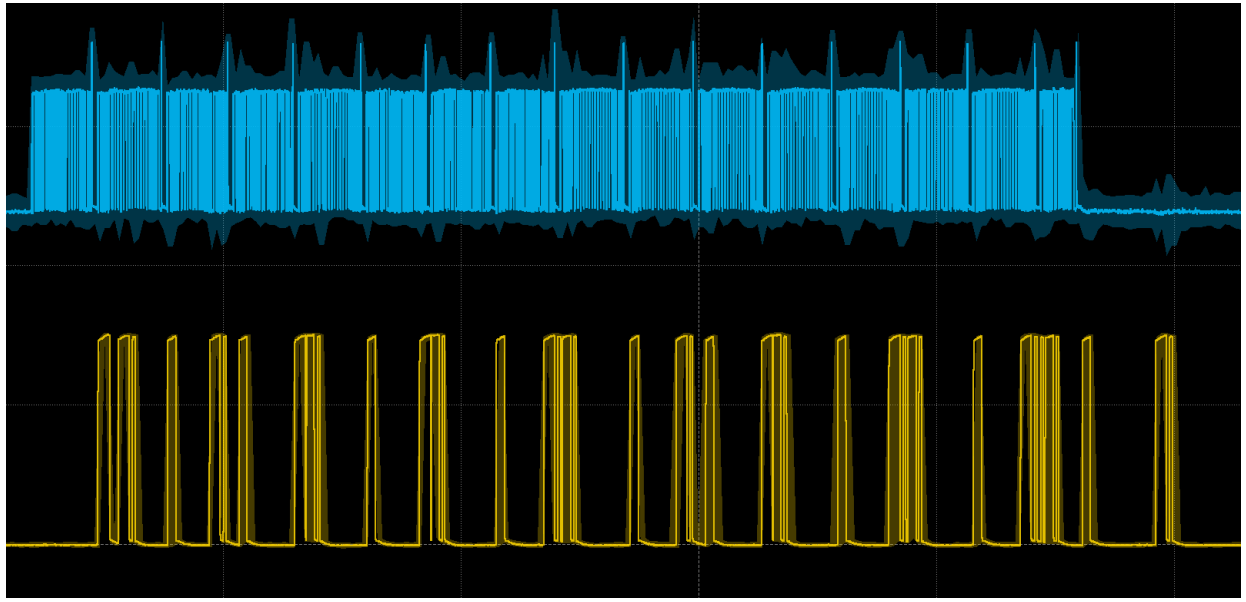
Using this base I've implemented PF_CAN and SOCK_RAW network families. It's mainly based on net/pkt since we're just sending raw packets



NuttX SocketCAN

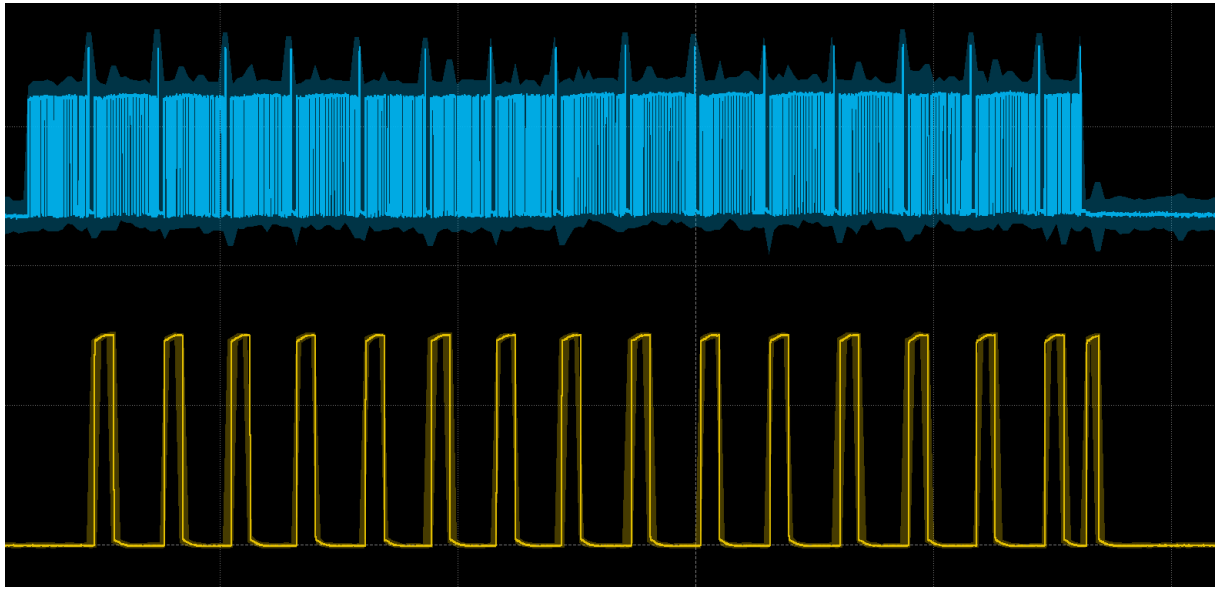
S32K1XX FlexCAN driver

- Re-based on the s32k1xx_enet driver
- Problem: Original Enet driver uses NuttX workqueue to process packets.
  Unfortunely the time (latency) before the workqueue gets executed can be too high which causes that RX mailboxes overflow thus we will drop CAN frames



S32K1XX FlexCAN driver

- Process CAN frame directly during an interrupt? Doesn't work because we need to able to get the network lock.
- Solution: net_trylock(), if the network isn't locked we take the lock and put the frame in the socket. If we don't get the lock store the frame in the read-ahead buffer. So that the RX mailbox can be released.



SocketCAN and real-time?

- For the UAVCAN use-case we needed SocketCAN extensions to improve real-time performance
- For receiving we've implemented the SO_TIMESTAMP sockopt in NuttX
  Which generates a timestamp for each incoming CAN frame
- Furthermore we needed TX deadline option so that transmitted frames have a deadline, when this deadline is met. The CAN driver has to remove the frame from it's TX mailbox.
  This option is called CAN_RAW_TX_DEADLINE

Control messages (CMSG) POSIX 1003.1g

- To extend the recvmsg() and sendmsg() with extra timestamping information we needed CMSG support
- The cmsghdr was already implemented in include/sys/socket.h, but recvmsg() & sendmsg() didn't support it yet.
- For now only SocketCAN supports CMSG, but in the future it would be interesting to add CMSG and SO_TIMESTAMP support to the Ethernet layer as well for time-sensitive use-cases

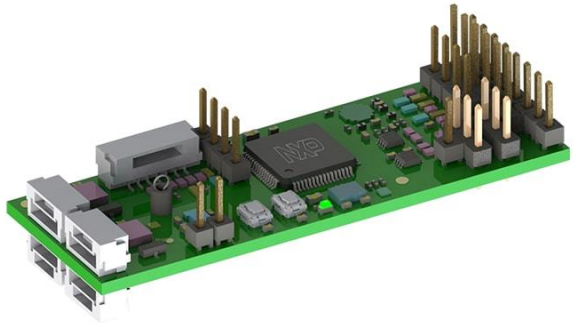
Porting a SocketCAN application to NuttX

- Use case: porting can-utils candump tool to NuttX
- Only have to change the headers, and disable the signals NuttX doesn't support
- Candump & cansend have included in NuttX-apps as CANUTILS_CANDUMP & CANUTILS_CANSEND

```
diff --git a/candump.c b/candump_nuttx.c
index 846c706..4044aac 100644
--- a/candump.c
+++ b/candump_nuttx.c
@@ -60,8 +60,8 @@
 #include <sys/uio.h>
 #include <net/if.h>

-#include <linux/can.h>
-#include <linux/can/raw.h>
+#include <nuttx/can.h>
+#include <netpacket/can.h>

 #include "terminal.h"
 #include "lib.h"
@@ -248,8 +248,10 @@ int main(int argc, char **argv)
        struct timeval timeout, timeout_config = { 0, 0 }, *timeout_current = NULL;
        FILE *logfile = NULL;

+#if 0 /* NuttX doesn't support these signals */
        signal(SIGTERM, sigterm);
        signal(SIGHUP, sigterm);
+#endif
        signal(SIGINT, sigterm);

        last_tv.tv_sec  = 0;
```

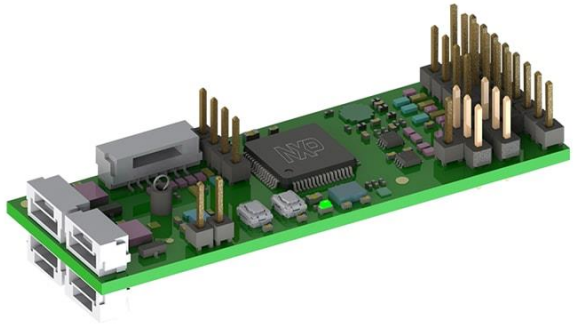
SocketCAN candump Demo

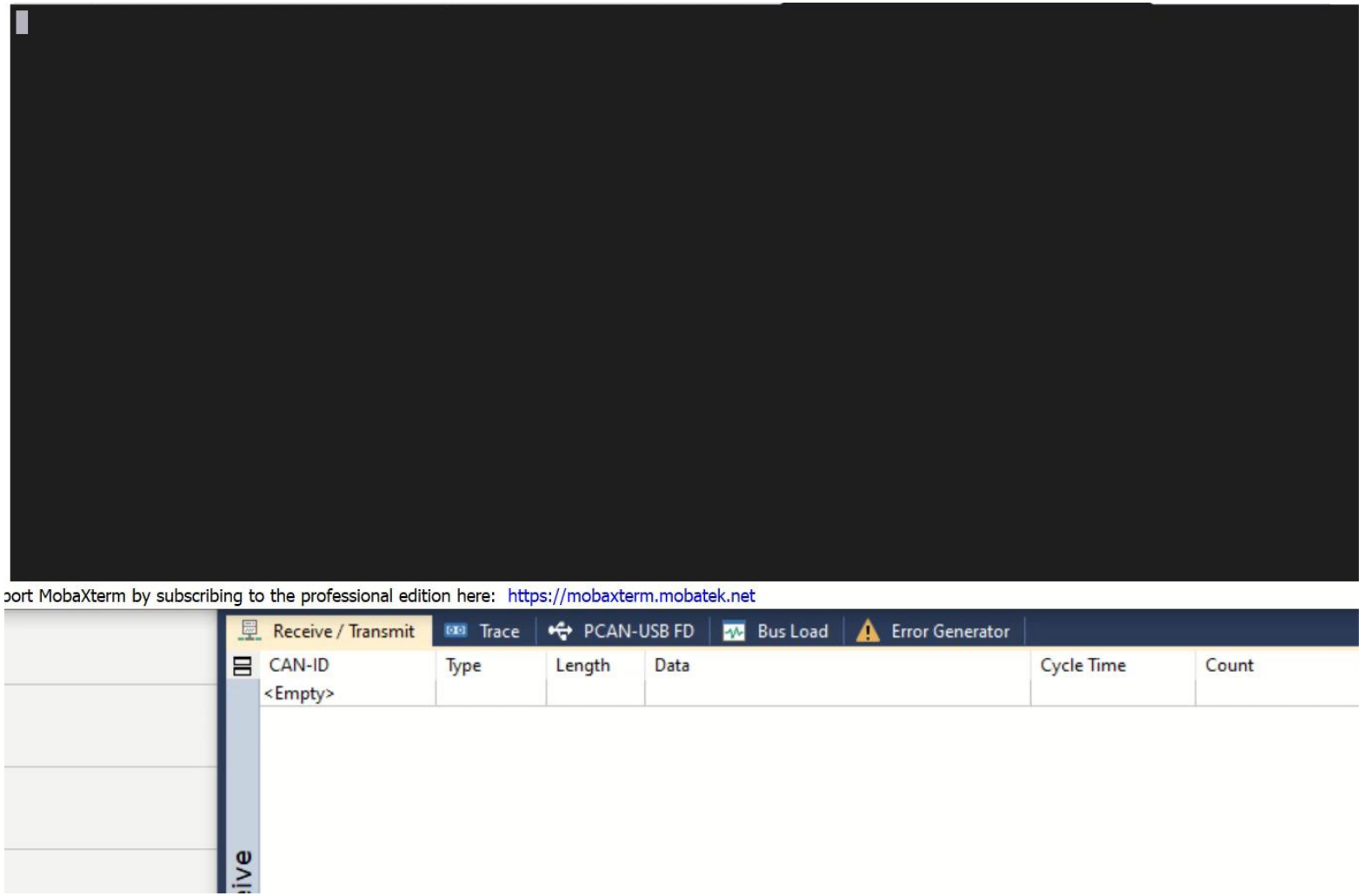
RDDRONE-UCANS32K146



SocketCAN cansend Demo

RDDRONE-UCANS32K146





Where do I use NuttX/SocketCAN for?

Being part of the Mobile robotics team in NXP, I'm currently focusing on reliable communication for small robotics (rovers, drones, etc)

Furthermore we're implementing UAVCAN v1 on top SocketCAN for use in the PX4 autopilot

Also my colleagues Jari & Cis also worked with SocketCAN/UAVCAN making a Smart BMS using NuttX
Which they will present on August 16 - 17 GMT

Conclusion

SocketCAN present a developer a standard POSIX socket based API to send/receive independent from the CAN hardware.

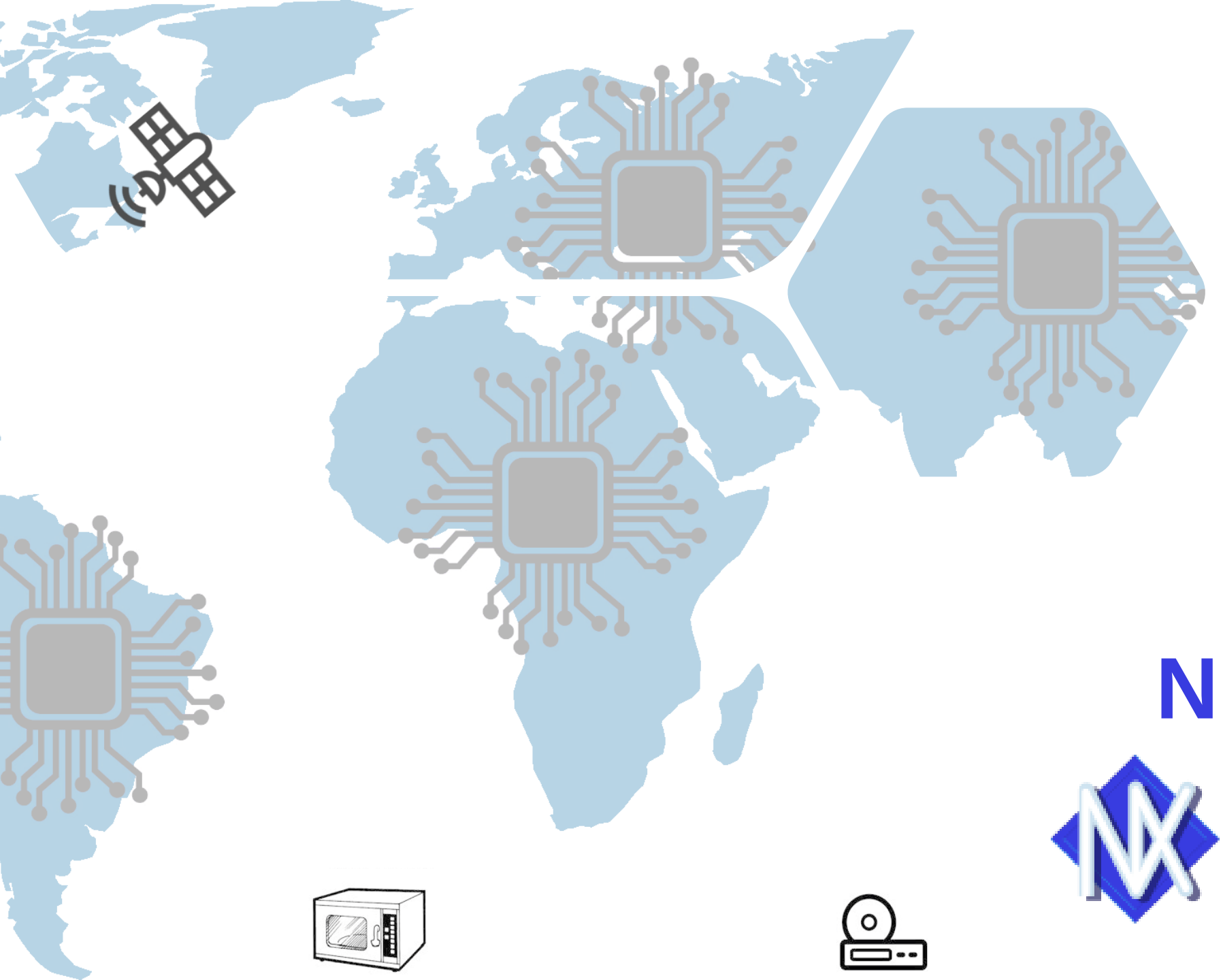
Current hardware supporting SocketCAN:

- NXP S32K1XX Family
- NXP Kinetis Family

In the future we also want to SocketCAN support to:

- NXP i.MX RT Family
- NXP LPC17xx_40xx Family

To add SocketCAN support to your favorite MCU you can find the instructions in the NuttX porting guide in chapter 6.3.10



Thank you!

NuttX Online Workshop