

# CANopen in the Shell

## Managing and Developing CANopen Applications on Linux

Thomas Flynn

September 28, 2017

Me (Tom Flynn):

- BEng. Mechatronics
- Linux User
- Industrial Electronics
- Engineer

# Introduction- Outline

## ① Controller Area Networks - What, Why, How?

# Introduction- Outline

- 1 Controller Area Networks - What, Why, How?
- 2 Peeking at Protocols

# Introduction- Outline

- 1 Controller Area Networks - What, Why, How?
- 2 Peeking at Protocols
- 3 CAN Support in the Kernel

# Introduction- Outline

- 1 Controller Area Networks - What, Why, How?
- 2 Peeking at Protocols
- 3 CAN Support in the Kernel
- 4 Cometh the Application Layer

# Introduction- Outline

- ① Controller Area Networks - What, Why, How?
- ② Peeking at Protocols
- ③ CAN Support in the Kernel
- ④ Cometh the Application Layer
- ⑤ The CANopen FOSS Tool Kit

# Introduction- Outline

- 1 Controller Area Networks - What, Why, How?
- 2 Peeking at Protocols
- 3 CAN Support in the Kernel
- 4 Cometh the Application Layer
- 5 The CANopen FOSS Tool Kit
- 6 Smooth and Unproblematic Demonstration



# Introduction- GNU Image Manipulation



# A Timeline:

- **1980s** Continuing increase of electronic sensors in vehicles. Signalling limited by environmental noise, complexity of wiring, limited I/O on controllers.

# A Timeline:

- **1980s** Continuing increase of electronic sensors in vehicles. Signalling limited by environmental noise, complexity of wiring, limited I/O on controllers.
- **1983** Development of the CAN bus concept at Robert Bosch GmbH

# A Timeline:

- **1980s** Continuing increase of electronic sensors in vehicles. Signalling limited by environmental noise, complexity of wiring, limited I/O on controllers.
- **1983** Development of the CAN bus concept at Robert Bosch GmbH
- **1986** CAN bus 'protocol' released at SAE conference.

# A Timeline:

- **1980s** Continuing increase of electronic sensors in vehicles. Signalling limited by environmental noise, complexity of wiring, limited I/O on controllers.
- **1983** Development of the CAN bus concept at Robert Bosch GmbH
- **1986** CAN bus 'protocol' released at SAE conference.
- **1987** First CAN transceiver chips released by Intel and Phillips

# A Timeline:

- **1980s** Continuing increase of electronic sensors in vehicles. Signalling limited by environmental noise, complexity of wiring, limited I/O on controllers.
- **1983** Development of the CAN bus concept at Robert Bosch GmbH
- **1986** CAN bus 'protocol' released at SAE conference.
- **1987** First CAN transceiver chips released by Intel and Phillips
- **1988** First Production vehicle to feature CAN-based system.

# A Timeline:

- **1980s** Continuing increase of electronic sensors in vehicles. Signalling limited by environmental noise, complexity of wiring, limited I/O on controllers.
- **1983** Development of the CAN bus concept at Robert Bosch GmbH
- **1986** CAN bus 'protocol' released at SAE conference.
- **1987** First CAN transceiver chips released by Intel and Phillips
- **1988** First Production vehicle to feature CAN-based system.
- **1993** ISO CAN standard released, ISO 11898

# A Timeline:

- **1980s** Continuing increase of electronic sensors in vehicles. Signalling limited by environmental noise, complexity of wiring, limited I/O on controllers.
- **1983** Development of the CAN bus concept at Robert Bosch GmbH
- **1986** CAN bus 'protocol' released at SAE conference.
- **1987** First CAN transceiver chips released by Intel and Phillips
- **1988** First Production vehicle to feature CAN-based system.
- **1993** ISO CAN standard released, ISO 11898
- **1995** CAN in Automation (CiA) release CiA 301, CANopen application layer and communication profile - *MembersOnly*.



# A Timeline:

- **1980s** Continuing increase of electronic sensors in vehicles. Signalling limited by environmental noise, complexity of wiring, limited I/O on controllers.
- **1983** Development of the CAN bus concept at Robert Bosch GmbH
- **1986** CAN bus 'protocol' released at SAE conference.
- **1987** First CAN transceiver chips released by Intel and Phillips
- **1988** First Production vehicle to feature CAN-based system.
- **1993** ISO CAN standard released, ISO 11898
- **1995** CAN in Automation (CiA) release CiA 301, CANopen application layer and communication profile - *MembersOnly*.
- **2011** CiA 301 V 4.2 made public - *Open*

# Controller Area Networks - What, Why, How?- Definition

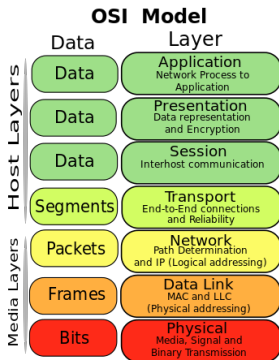
What is a Controller Area Network?

A network of nodes exchanging messages.

# Controller Area Networks - What, Why, How?- Technology Context

## OSI 7 Layer Model

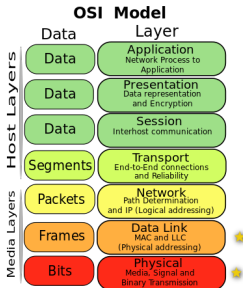
Observing the Open Standards Initiative (OSI) Model we can contextualise the aspects of CAN technology.



# Controller Area Networks - What, Why, How?- CAN as Foundation

## Layers 1 and 2

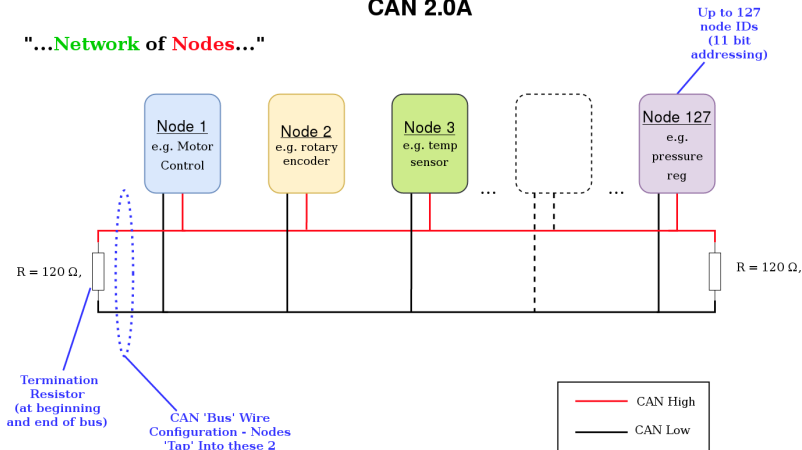
From the bottom up, a CAN network needs to be looked at in terms of its physical implementation. We will consider the Physical (L1) and Data Link (L2) layers.



# Controller Area Networks - What, Why, How?- Network

## Generic Network Topology CAN 2.0A

"...Network of Nodes..."

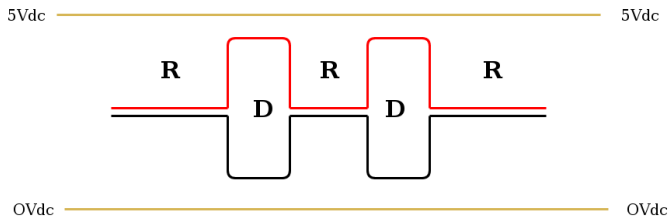


# Controller Area Networks - What, Why, How?- Exchange

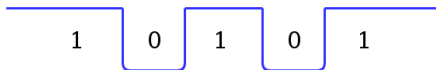
## CAN Signaling

D - Dominant - Both lines in 'on' state - Logical 0

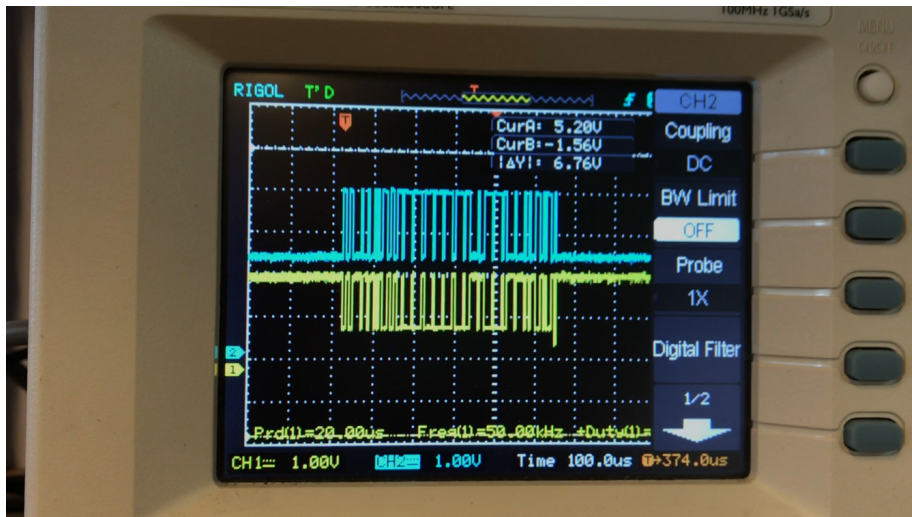
R - Recessive - Both lines in 'off' state - Logical 1



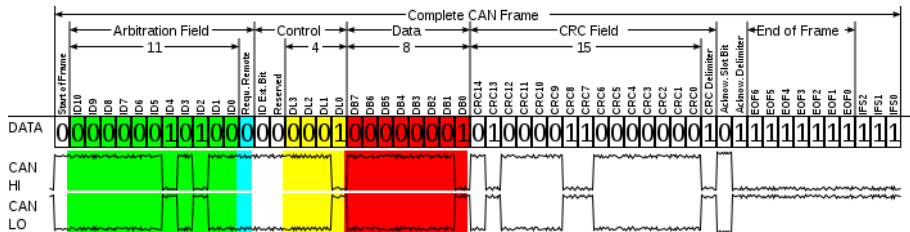
Resulting Data:



# Controller Area Networks - What, Why, How?- Signaling Example



# Controller Area Networks - What, Why, How?- Messages





# Controller Area Networks - What, Why, How?- Theory to Practice

To implement the hardware of a CAN-based device, we need to provide the L1 and L2 features.

# Controller Area Networks - What, Why, How?- Theory to Practice

To implement the hardware of a CAN-based device, we need to provide the L1 and L2 features.

Two Options:

- 1 Use a standalone IC that covers both L1 and L2 requirements *OR*

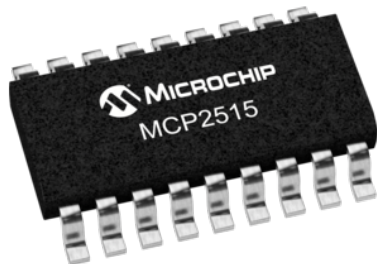
# Controller Area Networks - What, Why, How?- Theory to Practice

To implement the hardware of a CAN-based device, we need to provide the L1 and L2 features.

Two Options:

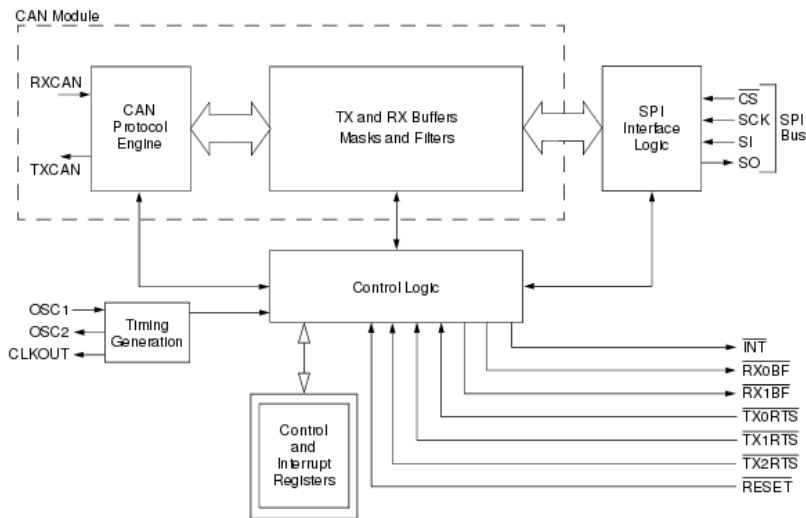
- 1 Use a standalone IC that covers both L1 and L2 requirements *OR*
- 2 Use an SoC/MCU with built in CAN module (L2) and possibly a transceiver IC (L1)

# Controller Area Networks - What, Why, How?- Example - Standalone IC

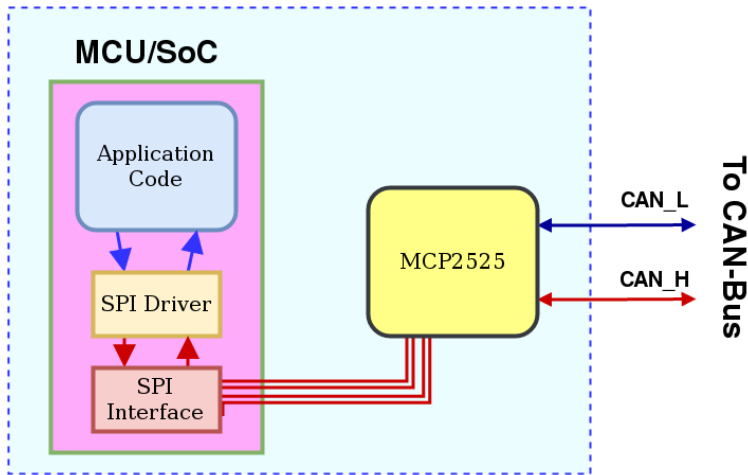


## Microchip MCP2515

# Controller Area Networks - What, Why, How?- Example - MCP2525



# Controller Area Networks - What, Why, How?- Hardware Solution with MCP2525



**Your CAN-Interfaced Device**

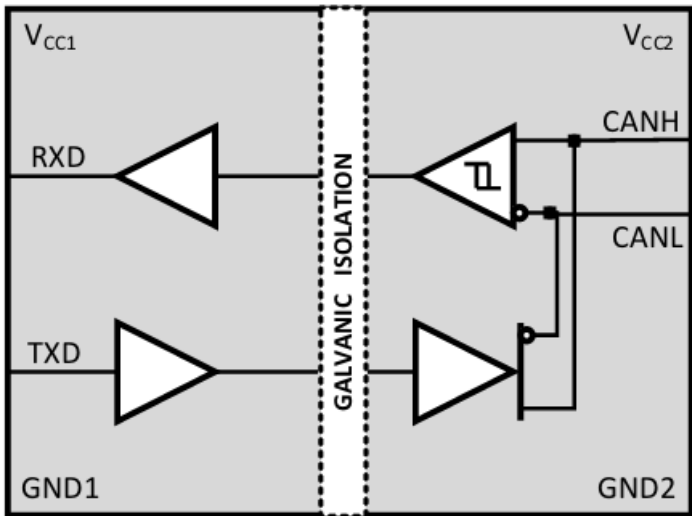
# Controller Area Networks - What, Why, How?- Example - Use Integrated L2 CAN with L1 Transceiver

## Texas Instruments ISO1050 Isolated CAN



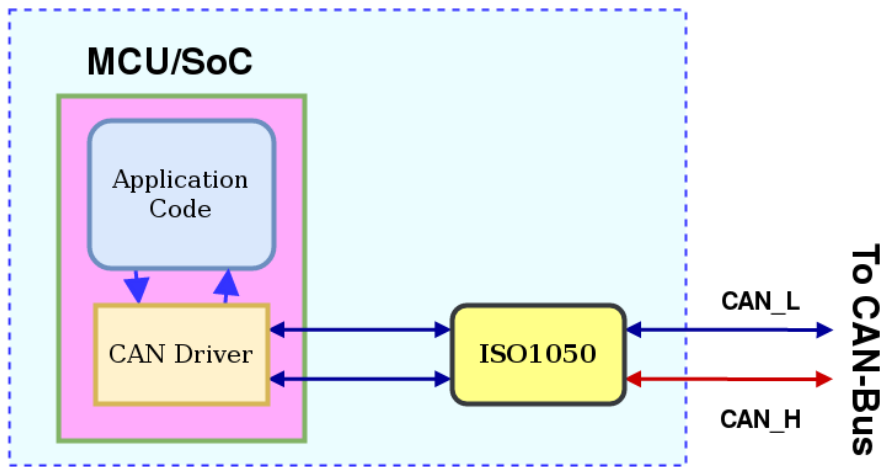
Transceiver

# Controller Area Networks - What, Why, How?- Example - TI ISO1050





# Controller Area Networks - What, Why, How?- Solution with ISO1050



**Your CAN-Interfaced Device**

# Peeking at Protocols- Moving up the Stack

**Q: What to do with all this reliable L1 and L2 infrastructure?**

# Peeking at Protocols- Moving up the Stack

**Q: What to do with all this reliable L1 and L2 infrastructure?**

A: Write standards of how our devices will use it!

# Peeking at Protocols- Horses for Courses

Some examples of the protocols using CAN as a foundation:

**J1939 1990** Control in heavy machinery e.g. Trucks, Tractors.  
Created and governed by SAE. Baud rate of 250kbit/s, up to 30 nodes.

# Peeking at Protocols- Horses for Courses

Some examples of the protocols using CAN as a foundation:

**J1939 1990** Control in heavy machinery e.g. Trucks, Tractors.  
Created and governed by SAE. Baud rate of 250kbit/s, up to 30 nodes.

**DeviceNet 1992** Industrial control. Created by Allen Bradley, eventually governed by ODVA. Baud rate of 125/250/500kbits/s, up to 64 nodes.

# Peeking at Protocols- Horses for Courses

Some examples of the protocols using CAN as a foundation:

**J1939 1990** Control in heavy machinery e.g. Trucks, Tractors.  
Created and governed by SAE. Baud rate of 250kbit/s, up to 30 nodes.

**DeviceNet 1992** Industrial control. Created by Allen Bradley, eventually governed by ODVA. Baud rate of 125/250/500kbits/s, up to 64 nodes.

**CANopen 1994** Industrial control. Created/governed by CiA. Up to 1Mbit/s, up to 127 nodes.

How come all these protocols?

How come all these protocols?

- Purpose



How come all these protocols?

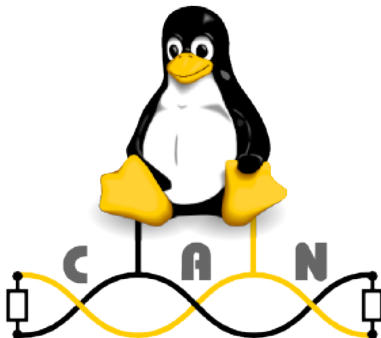
- Purpose
- Flexibility

How come all these protocols?

- Purpose
- Flexibility
- Implementation

**Road Block:** Before we can write application code (e.g. implement a protocol-compliant application), we need an API to talk to the CAN hardware/network.

**Road Block:** Before we can write application code (e.g. implement a protocol-compliant application), we need an API to talk to the CAN hardware/network.



## SocketCAN:

- **2008** 2.6.25 Linux Kernel - Patched with CAN support

## SocketCAN:

- **2008** 2.6.25 Linux Kernel - Patched with CAN support
- Known as Socket CAN (`linux\can\core.h`), contributed by Urs Thuermann, Oliver Hartkopp + More from Volkswagen Group Electronic Research

## SocketCAN:

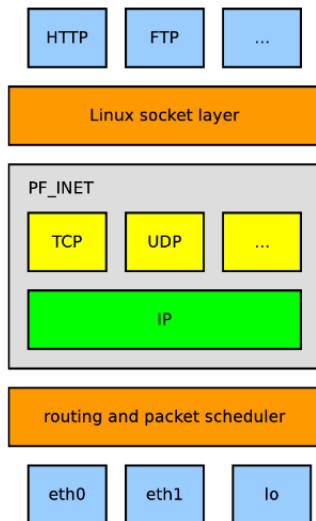
- **2008** 2.6.25 Linux Kernel - Patched with CAN support
- Known as Socket CAN (`linux\can\core.h`), contributed by Urs Thuermann, Oliver Hartkopp + More from Volkswagen Group Electronic Research
- Previous CAN drivers used 'Character Device Driver' - somewhat of a hack

## SocketCAN:

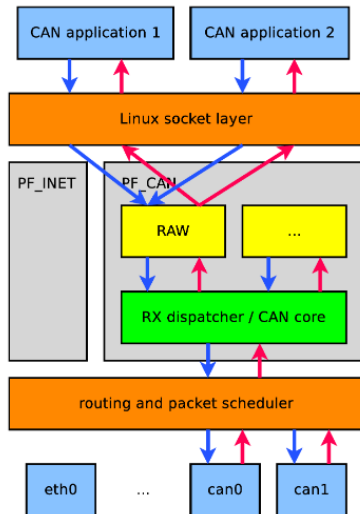
- **2008** 2.6.25 Linux Kernel - Patched with CAN support
- Known as Socket CAN (`linux\can\core.h`), contributed by Urs Thuermann, Oliver Hartkopp + More from Volkswagen Group Electronic Research
- Previous CAN drivers used 'Character Device Driver' - somewhat of a hack
- SocketCAN works within the Kernel's generic networking framework e.g. `#include sys\socket.h`, protocol family `PF_CAN`



# CAN in the Kernel- Generic Linux Network Stack



# CAN in the Kernel- PF\_CAN



# CANopen- CANopen Protocol

*CANopen provides several communication **objects**, which enable device designers to implement desired network behavior into a device. With these communication objects, device designers can offer devices that can communicate process data, indicate device-internal error conditions or influence and control the network behavior.*

- **CAN in Automation** description of CANopen

# CANopen- CANopen Protocol

Key concepts:

- 1 The **Object Dictionary** (OD)

# CANopen- CANopen Protocol

Key concepts:

- 1 The **Object Dictionary** (OD)
- 2 CAN Frames are called **messages**

# CANopen- CANopen Protocol

Key concepts:

- ① The **Object Dictionary** (OD)
- ② CAN Frames are called **messages**
- ③ Messages are **Communication Objects**

# CANopen- CANopen Protocol

Key concepts:

- ① The **Object Dictionary** (OD)
- ② CAN Frames are called **messages**
- ③ Messages are **Communication Objects**
- ④ Master+Slave / Client+Server functionality

# CANopen- CANopen Protocol

Key concepts:

- ① The **Object Dictionary** (OD)
- ② CAN Frames are called **messages**
- ③ Messages are **Communication Objects**
- ④ Master+Slave / Client+Server functionality
- ⑤ Network Management (**NMT**)  $\Rightarrow$  one node to rule them all