CAN BUS HACKING

Evandro Dessani Gomes e Renan Mantuanelli de Aquino
Universidade Federal do Espírito Santo (UFES)
Laboratório de Computação de Alto Desempenho (LCAD)

***Abstract* — This paper is about a car hacking study for reading and acting in an automotive Controller Area Network (CAN) Bus. For this purpose, was considered the Hybrid Ford Escape automobile, known as IARA (Intelligent Autonomous Robotic Automobile) belonging to the Auto Performance Computation Laboratory (LCAD) of the Universidade Federal do Espírito Santo (UFES). The results show that it is possible to read and send commands to actuate in the vehicle through the CAN bus in order to assist in the displacement of the autonomous vehicle.**

***Keywords* — Car hacking, Automotive CAN Bus.**

I. Introduction

The automotive industry has churned out some amazing vehicles, with complicated electronics and computer systems, but it has released little information about what makes those systems work. Understanding how vehicles communicate can seamlessly integrate other systems into a vehicle, like an additional display to show performance or can contribute to the development of autonomous vehicle.

The main goal of this paper is to build a hardware that is capable of reading and to send message to the CAN bus of the IARA vehicle (Intelligent Autonomous Robotic Automobile) that contribute to integrate with other systems which make the vehicle autonomous.

II. Literature Review

A. Controller Area Network

Controller area network (CAN) is a simple protocol used in manufacturing and in the automobile industry. Modern vehicles are full of little embedded systems and electronic control units (ECUs) that can communicate using the CAN protocol.

CAN has been a standard on US cars and light trucks since 1996, but it was not made mandatory until 2008 (2001 for European vehicles). If your car is older than 1996, it still may have CAN.

CAN runs on two wires: CAN high (CANH) and CAN low (CANL) and uses differential signaling, which means that when a signal comes in, CAN raises the voltage on one line and drops the other line an equal amount. Differential signaling is used in environments that must be fault tolerant to noise, such as in automotive systems and manufacturing.

Each CAN bus packet contains four key elements:

Arbitration ID: The arbitration ID is a broadcast message that identifies the ID of the device trying to communicate, though any one device can send multiple arbitration IDs. If two CAN packets are sent along the bus at the same time, the one with the lower arbitration ID wins.

Identifier extension (IDE): This bit is always zero (0) for standard CAN.

Data length code (DLC): This is the size of the data, which ranges from zero (0) to eight (8) bytes.

Data: This is the data itself. The maximum size of the data carried by a standard CAN bus packet can be up to 8 bytes, but some systems force 8 bytes by padding out the packet.

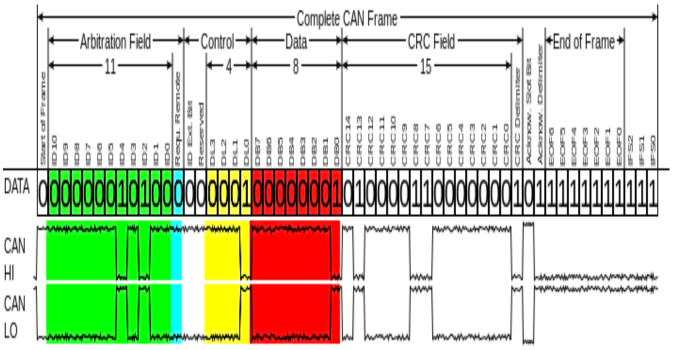


Fig. 1. Format of standard CAN packets.

B. Onboard Diagnostics

Many vehicles come equipped with an on board diagnostics (OBD-II) connector, also known as the diagnostic link connector (DLC), which communicates with the vehicle's internal network. The connector usually stay under the steering column or hidden elsewhere on the dash in a relatively accessible place.

CAN is easy to find when hunting through cables because its resting voltage is 2.5V. When a signal comes in, it will add or subtract 1V (3.5V or 1.5V). CAN wires run through the vehicle and connect between the ECUs and other sensors, and they are always in dual-wire pairs. You should find the CANH and CANL connections on pins 6 and 14 of your OBD-II connector.

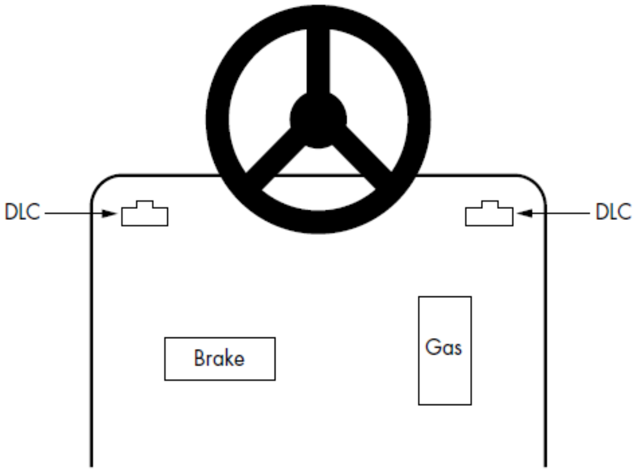


Fig. 2. DLC Location.

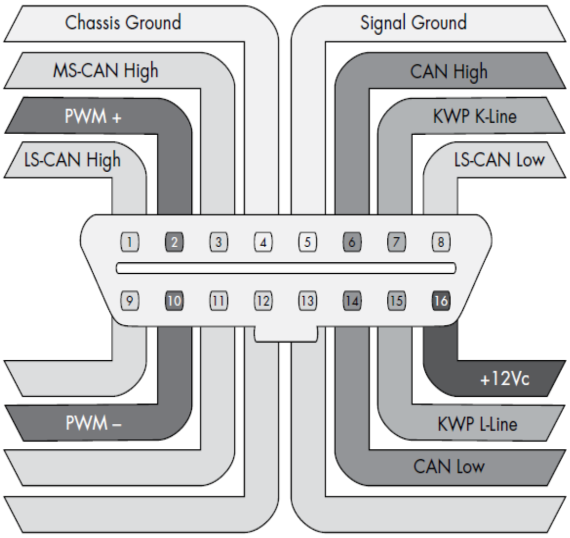


Fig. 3. Complete OBD II pinout connector

TABLE I
OBD II PINOUT

PIN	DESCRIPTION	PIN	DESCRIPTION
1	Vendor Option	9	Vendor Option
2	J1850 Bus +	10	j1850 BUS
3	Vendor Option	11	Vendor Option
4	Chassis Ground	12	Vendor Option
5	Signal Ground	13	Vendor Option
6	CAN (J-2234) High	14	CAN (J-2234) Low
7	ISO 9141-2 K-Line	15	ISO 9141-2 Low
8	Vendor Option	16	Battery Power

C. *Hardware Paspberry Pi3*

The Raspberry Pi provides a Linux operating system but does not include a CAN transceiver, so it is necessary to purchase a shield.

One of the advantages of using a Raspberry Pi over an Arduino is that it allows you to use the Linux Socket CAN tools directly, without the need to buy additional hardware. In general, a Raspberry Pi can talk to an MCP2515 over SPI with just some basic wiring.

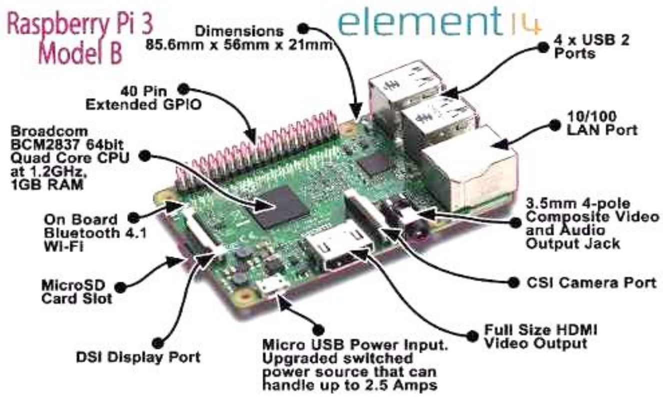
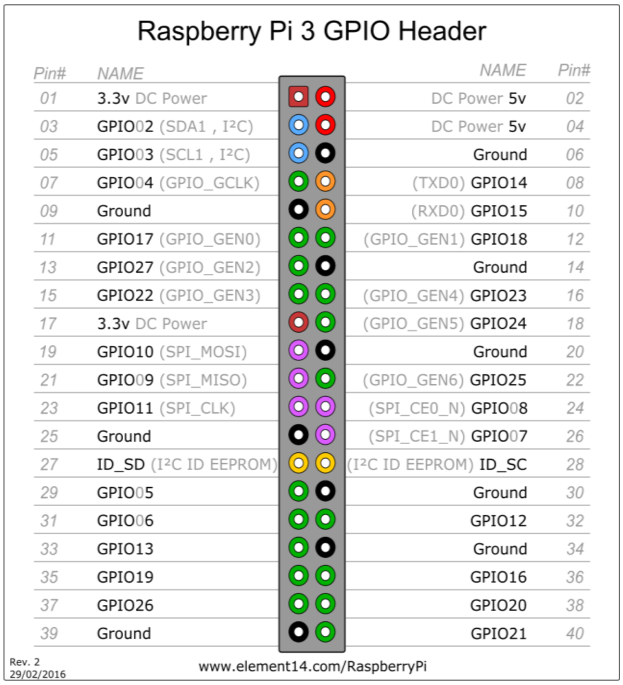


Fig. 4. Hardware Raspberry Pi 3


Raspberry Pi 3 GPIO Header

Pin#	NAME	NAME	Pin#
01	3.3v DC Power	DC Power 5v	02
03	GPIO02 (SDA1 , I²C)	DC Power 5v	04
05	GPIO03 (SCL1 , I²C)	Ground	06
07	GPIO04 (GPIO_GCLK)	(TXD0) GPIO14	08
09	Ground	(RXD0) GPIO15	10
11	GPIO17 (GPIO_GEN0)	(GPIO_GEN1) GPIO18	12
13	GPIO27 (GPIO_GEN2)	Ground	14
15	GPIO22 (GPIO_GEN3)	(GPIO_GEN4) GPIO23	16
17	3.3v DC Power	(GPIO_GEN5) GPIO24	18
19	GPIO10 (SPI_MOSI)	Ground	20
21	GPIO09 (SPI_MISO)	(GPIO_GEN6) GPIO25	22
23	GPIO11 (SPI_CLK)	(SPI_CE0_N) GPIO08	24
25	Ground	(SPI_CE1_N) GPIO07	26
27	ID_SD (I²C ID EEPROM)	(I²C ID EEPROM) ID_SC	28
29	GPIO05	Ground	30
31	GPIO06	GPIO12	32
33	GPIO13	Ground	34
35	GPIO19	GPIO16	36
37	GPIO26	GPIO20	38
39	Ground	GPIO21	40

Rev. 2
29/02/2016
www.element14.com/RaspberryPi

Fig. 5. Raspberry Pi 3 Pinout

D. *Hardware Microchip MCP2515*

The Microchip Technology's MCP2515 is a stand-alone Controller Area Network (CAN) controller that implements the CAN specification.

It is capable of transmitting and receiving both standard and extended data and remote frames.

The MCP2515 has two acceptance masks and six acceptance filters that are used to filter out unwanted messages, thereby reducing the host MCUs overhead. The MCP2515 interfaces with microcontrollers (MCUs) via an industry standard Serial Peripheral Interface (SPI).

Fig. 6. Hardware MCP2515

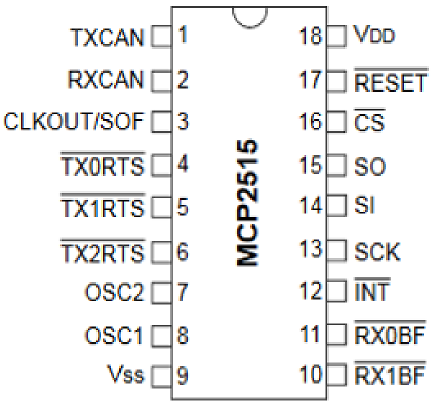


Fig. 7. MCP2515 Pinout

TABLE II
MCP2515 PINOUT AND FUNCTIONS

Name	PDIP/SOIC Pin #	TSSOP Pin #	I/O/P Type	Description	Alternate Pin Function
TXCAN	1	1	O	Transmit output pin to CAN bus	—
RXCAN	2	2	I	Receive input pin from CAN bus	—
CLKOUT	3	3	O	Clock output pin with programmable prescaler	Start-of-Frame signal
TX0RTS	4	4	I	Transmit buffer TXB0 request-to-send. 100 kΩ internal pull-up to VDD	General purpose digital input. 100 kΩ internal pull-up to VDD
TX1RTS	5	5	I	Transmit buffer TXB1 request-to-send. 100 kΩ internal pull-up to VDD	General purpose digital input. 100 kΩ internal pull-up to VDD
TX2RTS	6	7	I	Transmit buffer TXB2 request-to-send. 100 kΩ internal pull-up to VDD	General purpose digital input. 100 kΩ internal pull-up to VDD
OSC2	7	8	O	Oscillator output	—
OSC1	8	9	I	Oscillator input	External clock input
VSS	9	10	P	Ground reference for logic and I/O pins	—
RX1BF	10	11	O	Receive buffer RXB1 interrupt pin or general purpose digital output	General purpose digital output
RX0BF	11	12	O	Receive buffer RXB0 interrupt pin or general purpose digital output	General purpose digital output
INT	12	13	O	Interrupt output pin	—
SCK	13	14	I	Clock input pin for SPI interface	—
SI	14	16	I	Data input pin for SPI interface	—
SO	15	17	O	Data output pin for SPI interface	—
CS	16	18	I	Chip select input pin for SPI interface	—
RESET	17	19	I	Active low device reset input	—
VDD	18	20	P	Positive supply for logic and I/O pins	—
NC	—	6,15	—	No internal connection	

E. *Hardware Logic Gate 74HC00*

The MM74HC00 NAND gates utilize advanced silicon-gate CMOS technology to achieve operating speeds similar to LS-TTL gates with the low power consumption of standard CMOS integrated circuits.

All devices have high noise immunity and the ability to drive 10 LS-TTL loads. The 74HC logic family is functionally as well as pin-out compatible with the standard 74LS logic family. All inputs are protected from damage due to static discharge by internal diode clamps to VCC and ground.

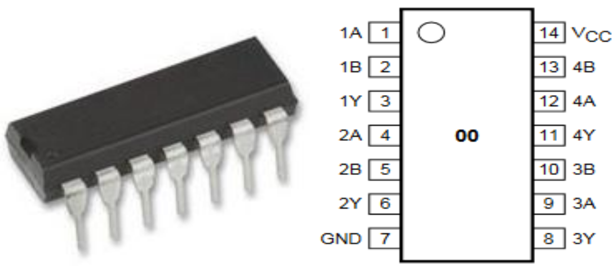


Fig. 8. Logic gate 74HC00

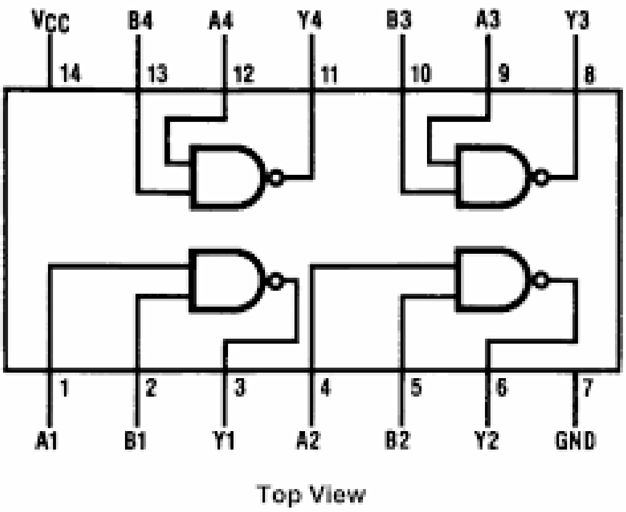


Fig. 9. 74HC00 pinout

TABLE III
74HC00 PINOUT AND FUNCTIONS

PIN	SYMBOL	DESCRIPTION
1	1A	data input
2	1B	data input
3	1Y	data output
4	2A	data input
5	2B	data input
6	2Y	data output
7	GND	ground (0 V)
8	3Y	data output
9	3A	data input
10	3B	data input
11	4Y	data output
12	4A	data input
13	4B	data input
14	V_{CC}	supply voltage

F. *Software Raspbian for Raspberry Pi3*

Raspbian is a free operating system based on Debian optimized for the Raspberry Pi hardware. An operating system is the set of basic programs and utilities that make your Raspberry Pi run. However, Raspbian provides more than a pure OS: it comes with over 35,000 packages, pre-compiled software bundled in a nice format for easy installation on your Raspberry Pi.

The initial build of over 35,000 Raspbian packages, optimized for best performance on the Raspberry Pi, was completed in June of 2012. However, Raspbian is still under active development with an emphasis on improving the stability and performance of as many Debian packages as possible.

III. Material and Methods

A. *Circuit Assembly*

To assemble the circuit It is necessary the components below:

- 1 Raspberry Pi 3;
- 2 MCP2515;
- 1 74HC00;
- 1 resistor: 1k ohms;
- 1 resistor: 2K ohms;
- Wires;

- 1 case;
- 1 Soldering iron;
- 1 Tin soldering wire;
- Screws;

The electrical diagram must be assembly as below:

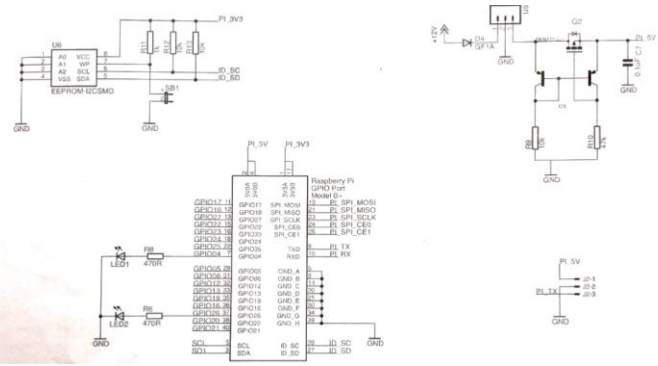


Fig. 10. Electrical diagram for raspberry pi3

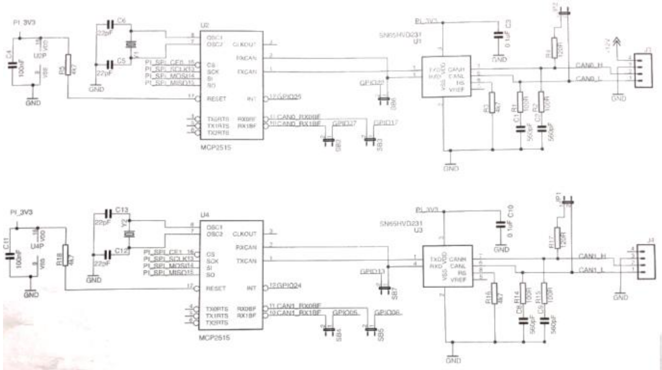


Fig. 11. Electrical diagram for MCP2515

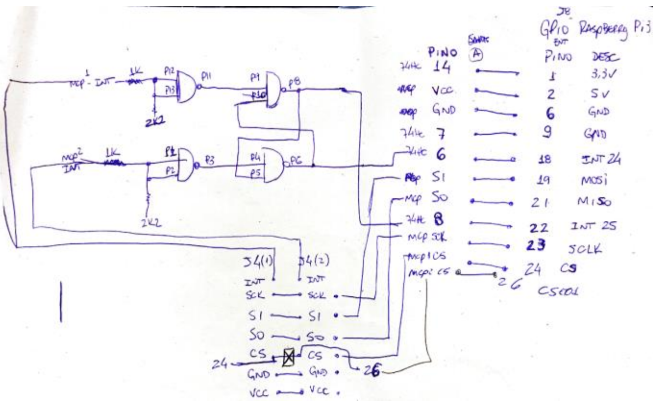


Fig. 12. Electrical diagram for MCP2515

Fig. 13. Case with all hardware integrated

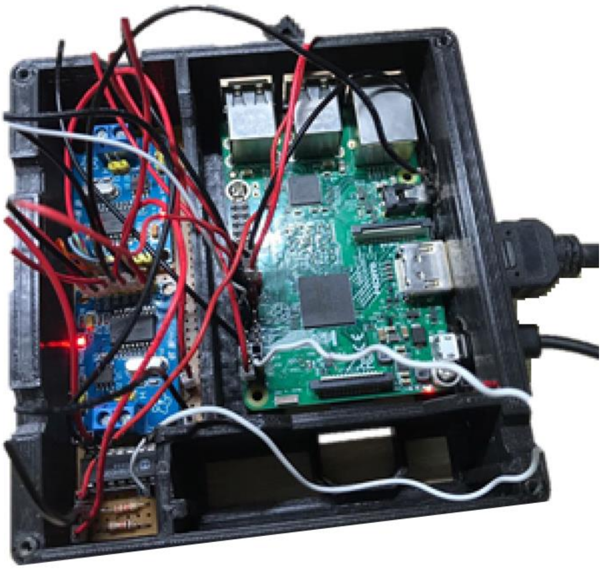


Fig. 14. Case with all hardware integrated

B. *Rasp Configuration*

To configure the raspberry Pi 3 It is necessary the items below:

- 1 HDMI cable;
- 1 Micro USB cable (3.3 V power supply);
- 1 Keyboard;
- 1 mouse;
- Internet connection;

To configure just follow the steps below:

- Download the Raspbian Jessie Lite
  https://www.raspberrypi.org/downloads/
- Save as image in the micro SD card (bigger than 4GB);
- Enable SPI:
  - sudo nano /boot/config.txt;
  - Remove comment (#) from the following line: dtparam=spi=on
  - Insert the lines below:
    - dtoverlay=mcp2515-can0,oscillator=8000000,interrupt=25
    - dtoverlay=mcp2515-can1,oscillator=8000000,interrupt=24
    - dtoverlay=spi-bcm2835-overlay

      An oscillator speed is specified by MCP2515 as shown below

Fig. 15. Oscillator specification on MCP2515

- Download Can Utils
  - sudo apt-get install can-utils

- Create file Rules.bat
    - sudo nano /rules.bat
    - cangw –F
    - cangw –A –s can1 –d can0 –f 0:0
    - cangw –A –s can0 –d can1 –f 425~1FFFFFFF
    - cangw –A –s can0 –d can1 –f 425:1FFFFFFF –m AND:ILD:425.8.00FFFFFFFFFFFFFF
- Create file Set.bat
    - sudo nano /set.bat
    - sudo /sbin/ip link set can0 up type can bitrate 500000
    - sudo /sbin/ip link set can1 up type can bitrate 500000
- Assign special permission to file Rules.bat
    - sudo chmod +x /rules.bat
- Edit file rc.local
    - Sudo nano /etc/rc.local
    - Insert the lines below:
        - /rules.bat
        - /set.bat
- Reboot the Raspbian
    - Sudo reboot

C. CAN Utilities Suite

There are many CAN utilities suite and very useful listed below:

- **Asc2log**

This tool parses ASCII CAN dumps in the following form into a standard SocketCAN logfile format: 0.002367 1 390x Rx d 8 17 00 14 00 C0 00 08 00

- **bcmserver**

Jan-Niklas Meier's proof-of-concept (PoC) broadcast manager server takes commands like the following: vcan1 A 1 0 123 8 11 22 33 44 55 66 77 88

- **canbusload**

This tool determines which ID is most responsible for putting the most traffic on the bus and takes the following arguments: *interface@bitrate*

- **can-calc-bit-timing**

This command calculates the bit rate and the appropriate register values for each CAN chipset supported by the kernel.

- **candump**

This utility dumps CAN packets. It can also take filters and log packets.

- **Canfdtest**

This tool performs send and receive tests over two CAN buses.

- **Cangen**

This command generates CAN packets and can transmit them at set intervals. It can also generate random packets.

- **loopback on**

This command send and receive its own CAN packets

- **cangw**

This tool manages gateways between different CAN buses and can also filter and modify packets before forwarding them on to the next bus.

- **canlogserver**

This utility listens on port 28700 (by default) for CAN packets and logs them in standard format to stdout.

- **canplayer**

This command replays packets saved in the standard SocketCAN "compact" format.

- **cansend**

This tool sends a single CAN frame to the network.

- **cansniffer**

This interactive sniffer groups packets by ID and highlights changed bytes.

- **isotpdump**

This tool dumps ISO-TP CAN packets

- **isotprecv**

This utility receives ISO-TP CAN packets and outputs to stdout.

- **isotpsend**

This command sends ISO-TP CAN packets that are piped in from stdin.

- **isotpserver**

This tool implements TCP/IP bridging to ISO-TP and accepts data packets in the format *1122334455667788*.

- **isotpsniffer**

This interactive sniffer is like cansniffer but designed for ISO-TP packets.

- **Isotptun**

This utility creates a network tunnel over the CAN network.

- **log2asc**

This tool converts from standard compact format to the following ASCII format: 0.002367 1 390x Rx d 8 17 00 14 00 C0 00 08 00

- **log2long**

This command converts from standard compact format to a user readable format.

- **slcan_attach**

This is a command line tool for serial-line CAN devices.

- **slcand**

This daemon handles serial-line CAN devices.

- **slcanpty**

This tool creates a Linux psuedoterminal interface (PTY) to communicate with a serial-based CAN interface.

D. Hybrid Ford Escape CAN Bus

The Hybrid Ford Escape has two CAN buses, a medium speed (MS) CAN bus operating at 125kbps and a high speed (HS) CAN bus operating at 500kbps. Both of these buses terminate at the OBD-II port, referred to in the Ford wiring diagrams as the Data Link Connector (DLC).

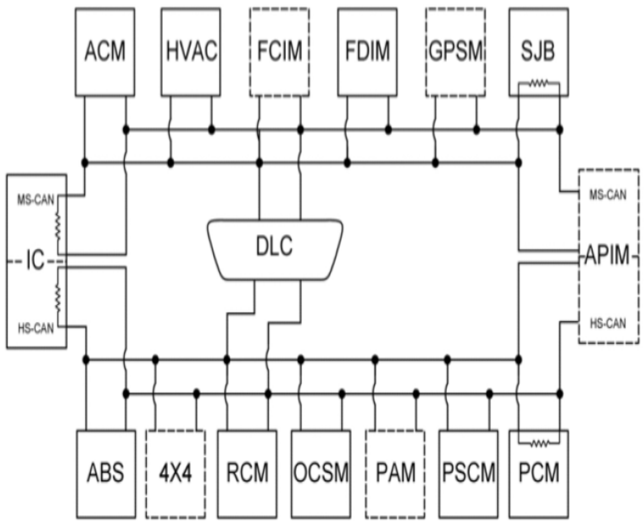


Fig. 16. Hybrid Ford Escape Controllers

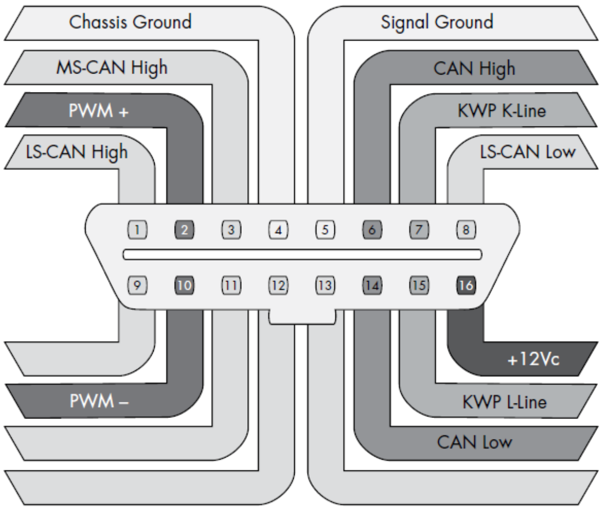


Fig. 3. Complete OBD II pinout connector

HS CAN (Pins 6 and 14):

1. Instrument Cluster (IC)
2. Anti-Lock Brake System Module (ABS)
3. Restraints Control Module (RCM)
4. Occupant Classification Module (OCSM)
5. Parking Aid Module (PAM)
6. Power Steering Control Module (PSCM)
7. Powertrain Control Module (PCM)
8. Accessory Protocol Interface Module (APIM)

MS CAN (Pins 3 and 11):

1. Instrument Cluster (IC)
2. Audio Control Module (ACM)
3. HVAC Module (HVAC)
4. Front Controls Interface Module (FCIM)
5. Front Display Module (FDM)
6. Smart Junction Box (SJB)
7. Accessory Protocol Interface Module (APIM)

In addition, below are listed the SAE standard and Ford Escape DTC's code.

- P0100–P0199: Fuel and air metering, auxiliary emissions controls
- P0100–P0199: Fuel and air metering
- P0200–P0299: Fuel and air metering (injector circuit)
- P0300–P0399: Ignition system or misfire
- P0400–P0499: Auxiliary emissions controls
- P0500–P0599: Vehicle speed controls, and idle control systems
- P0600–P0699: Computer output circuit
- P0700–P0799: Transmission

Besides these, the fault codes for each ECU are listed below:

- FordECU[0x0701]="GPSM"
- FordECU[0x0720]="IC"
- FordECU[0x0726]="SJB"
- FordECU[0x0727]="ACM"
- FordECU[0x0730]="PSCM"
- FordECU[0x0733]="HVAC"
- FordECU[0x0736]="PAM"
- FordECU[0x0737]="RCM"
- FordECU[0x0760]="ABS"
- FordECU[0x0761]="4x4"
- FordECU[0x0765]="OCSM"
- FordECU[0x07A6]="FDIM"
- FordECU[0x07A7]="FCIM"
- FordECU[0x07D0]="APIM"
- FordECU[0x07E0]="PCM"

IV. Results

After assembly of the electrical circuit and software configuration, it was possible to install the solution in the vehicle and identify the ID's through reverse engineering using the following commands as listed below:

- candump can0
- cansniffer can0
- cansniffer –c can 0

```
0.000000  110  00 00 00 00 00 00 00 00 ........
0.000000  120  F2 89 63 20 03 20 03 20 ..c . .
0.202675  128  A1 00 02                ...
0.000000  130  00 00 80 7E 00          ...~.
9.999999  131  36 46 31 30 39 31 34 39 6F109149
0.000000  170  01 00 00 00 00 00 00 00 ........
0.000000  300  00 00 84 00 00 04 00 00 ........
0.000000  308  00 4D 00 00 00 00 00 00 .M......
0.000000  320  20 04 00 00 00 00 00 00  .......
0.000000  348  00 00 00 00 00 00 00 00 ........
0.202618  380  02 02 00 00 E0 00 7F 0D ........
^C000000  388  01 10                   ..
0.000000  410  20 00 00 00 00 00 00 00  .......
0.000000  510  34 6F 01 3C F0 C4 12 6F 4o.<...o
0.000000  520  00 00 04 00 00 00 00 00 ........
9.999999  670  47 31 5A 54 35 33 38 32 G1ZT5382
```

Fig. 17. CAN Bus reading

One advantage of command cansniffer is that it is possible to send it keyboard input to filter results as they are displayed in the terminal.

For example, to see only IDs 131 and 670 as cansniffer collects packets, enter this:

-000000
+131
+670

Entering -000000 turns off all packets, and entering +131 and +670 filters out all except IDs 131 and 670.

The -000000 command uses a bitmask, which does a bit-level comparison against the arbitration ID.

V. ANALYSIS

The Ford escape has an electric steering that is driven by a small electric motor, which comes into action when receiving signals from sensors that identify the movement of the steering wheel and relieves the work of the driver's muscles. The exerted force varies depending on the speed of the vehicle, in other words, it applies more force when in maneuvers and less at high speed, leaving the car more "in the hand" of the driver.

In addition, the vehicle IARA comes with this default setting so that difficulties to operate autonomously because the vehicle does not realize right the angle on the steering wheel required when making curves during travel.

As a result, reverse engineering is one option to identify the ID of the steering wheel when the vehicle is at high speed. In addition, it was possible to send a command of speed constant equal to zero (0) through the CAN bus and the small electrical motor of the electric steering applies more force leaving the steering wheel “free” to perform curves at high speed in autonomous mode.

Finally, the commands to solve this problem are listed below as previously mentioned in this paper:

- Create file Rules.bat
    - sudo nano /rules.bat
    - cangw –F
    - cangw –A –s can1 –d can0 –f 0:0
    - cangw –A –s can0 –d can1 –f 425~1FFFFFFF
    - cangw –A –s can0 –d can1 –f 425:1FFFFFFF –m AND:ILD:425.8.00FFFFFFFFFFFFFF
- Create file Set.bat
    - sudo nano /set.bat
    - sudo /sbin/ip link set can0 up type can bitrate 500000
    - sudo /sbin/ip link set can1 up type can bitrate 500000
- Assign special permission to file Rules.bat
    - sudo chmod +x /rules.bat
- Edit file rc.local
    - Sudo nano /etc/rc.local
    - Insert the lines below:
        - /rules.bat
        - /set.bat
- Reboot the Raspbian
    - Sudo reboot

VI. CONCLUSIONS

Today, with the auto industry on the cusp of fully autonomous vehicle technology and greater interconnectivity than ever before, the CAN bus hacking is extremely important and necessary to solve problems and improve the performance of the vehicle when operated on autonomous mode.

REFERÊNCIAS

[1] ED, S. A Field Guide to Automotive Technology. [s.n.], 2009. 207 p.

[2] FRAUNHOFER-GESELLSCHAFT. Safer Car Controls. IEEE Vehicle Power and Propulsion Conference (VPPC), 2007.

[3] LI, R.; LIU, C.; LUO, F. A Design for Automotive CAN Bus Monitoring System. Science Daily, 2008.

[4] LINUX-CAN. can-utils:Linux-CAN/SocketCAN user space applications. 2017.

[5] NATALE, M. D. Understanding and using the Controller Area Network. [s.n.], 2009. 47 p.

[6] SMITH, C. The car hacker’s handbook. [s.n.], 2016. 304 p.