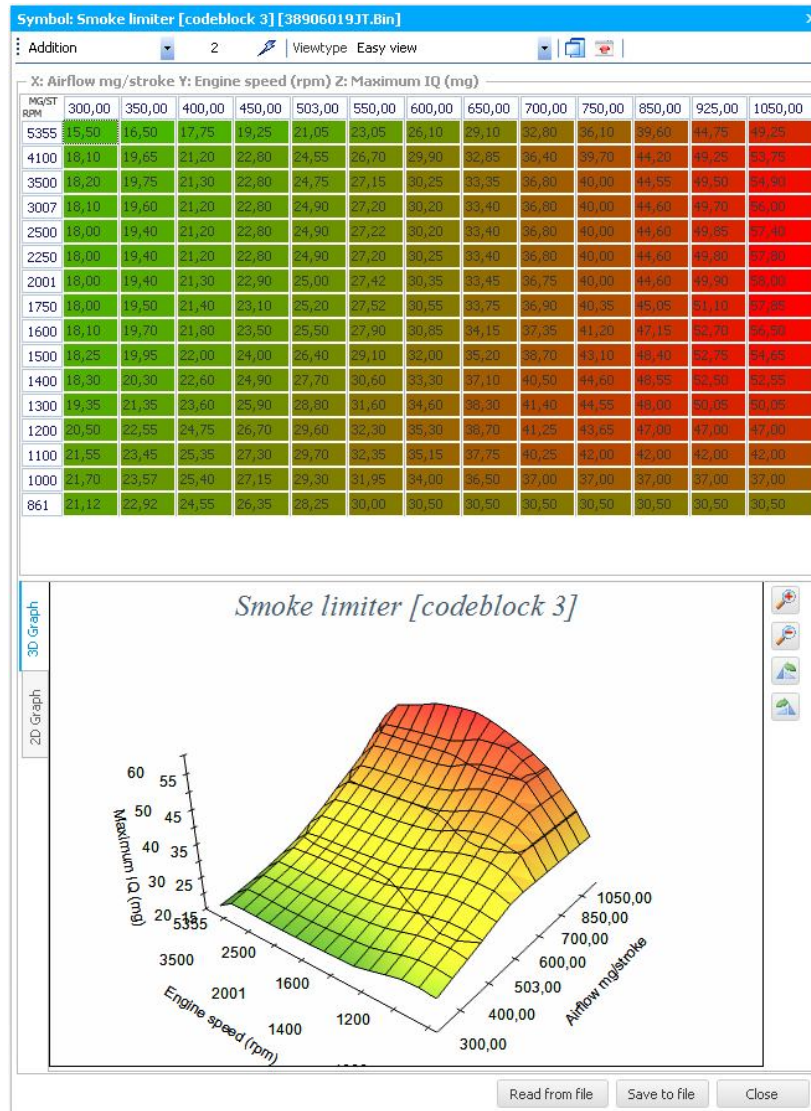


# VAG EDC15P

A detailed description for Bosch EDC15P ECUs used in VAG cars



## Table of content

Introduction .....	- 3 -
Hardware .....	- 4 -
Overview of the board .....	- 4 -
Main CPU: Infineon C167 .....	- 5 -
Flash eprom: Am29F400 .....	- 6 -
ECU Pinout .....	- 7 -
Software.....	- 9 -
Software information data.....	- 11 -
Reading the code.....	- 11 -
Maps and variables.....	- 12 -
Getting the map addresses .....	- 12 -
Validating the entries in the collection.....	- 12 -
Labelling the maps.....	- 13 -
Tuning EDC15P .....	- 14 -
Communication with the ECU .....	- 20 -
Connection diagram .....	- 20 -
KWP1281.....	- 20 -
Wakeup procedure for normal mode.....	- 20 -
Appendix I: Building a high speed K-line interface.....	- 21 -
Schematic.....	- 21 -
Parts information .....	- 21 -

## Introduction

The Bosch EDCxx series (Electronic Diesel Control) ECUs is a widely used system for modern diesel engines. It is used by BMW, VAG, Opel, SAAB and many others. This document and the described VAG EDC15P suite software will only focus on the Volkswagen Audi Gruppe (VAG) specific implementations in the EDC15P models. These ECUs are used in the PD engines (Pumpe Duse) from Volkswagen, Audi, Skoda and Seat. Other EDCxx ECUs may work in a similar fashion but will differ in certain areas.

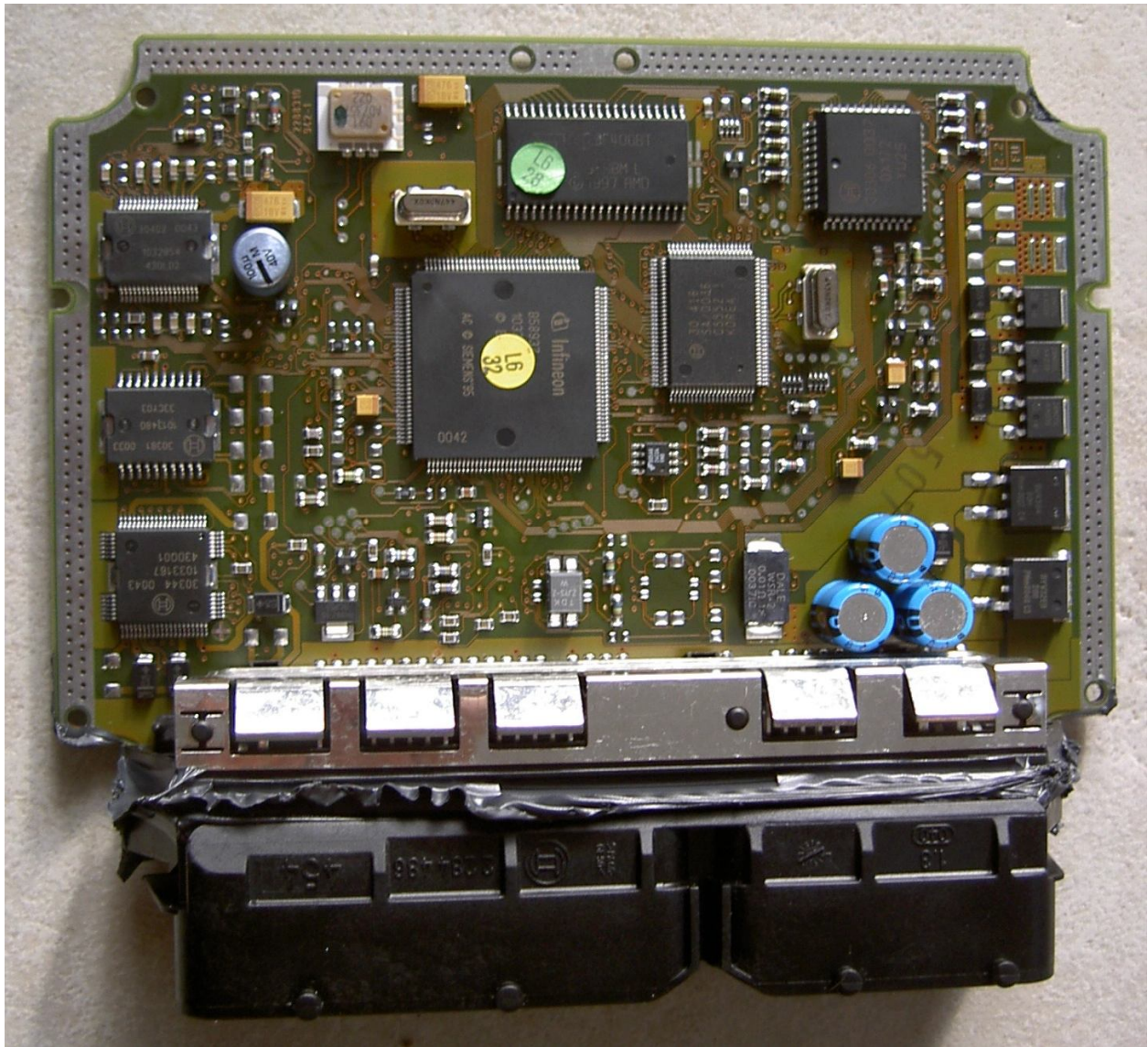
The document describes the EDC15P ECU in detail. It will first describe the hardware and proceed with a even more detailed description of the software that is running in the ECU so that we can learn how to tweak and tune the ECU to match the hardware – altered or not – that is on the car better.

*Special thanks for getting all this together go out to rkam, mtx-electronics, Pixis5 and others on [ecuconnections.com](http://ecuconnections.com) and [chiptuners.org](http://chiptuners.org).*

## Hardware

### Overview of the board

The ECU contains a multi-layer printed circuit board (PCB) which holds a lot of SMD components. The main components are – logically: Main CPU, Flash program storage, SRAM memory (working memory), EEPROM (for storing mileage, immo etc) and a lot of input/output (I/O).



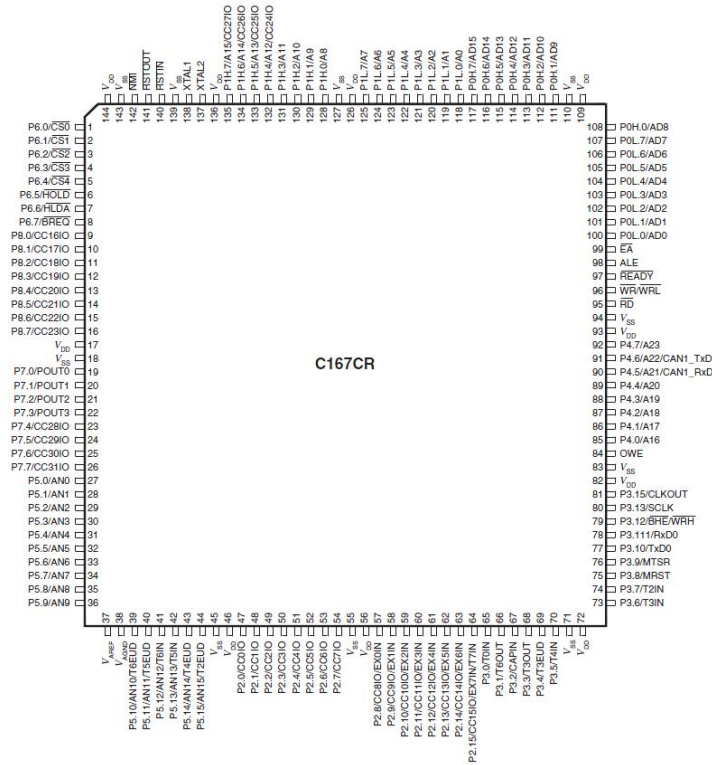
# Main CPU: Infineon C167

✓ 16 bit CPU

Datasheet documents

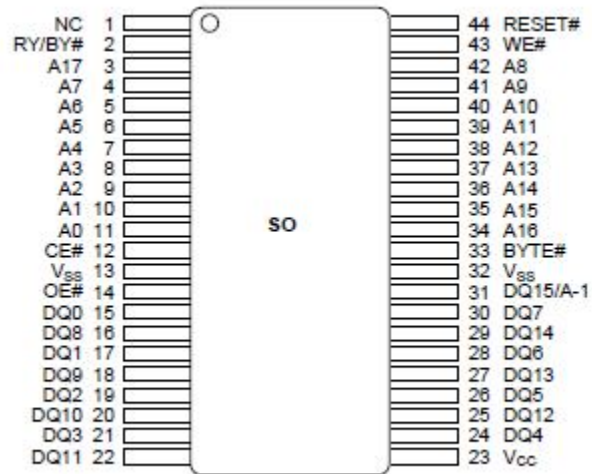
[http://trionic.mobixs.eu/EDC15P/c167cr\\_ds.pdf](http://trionic.mobixs.eu/EDC15P/c167cr_ds.pdf)

[http://trionic.mobixs.eu/EDC15P/c167cr\\_um.pdf](http://trionic.mobixs.eu/EDC15P/c167cr_um.pdf)

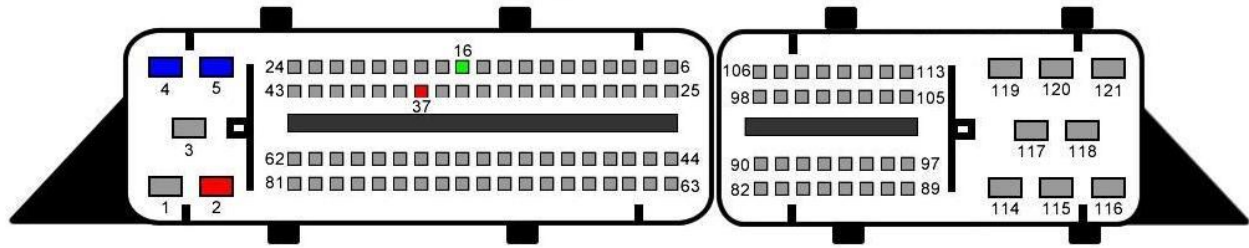


## Flash eprom: Am29F400

The flash contains the program and maps for the ECU. EDC15P has 512KB (4 Mbit) flash memory in which it can store multiple map segments for different situations (automatic gearbox, manual gearbox, quattro etc). Switching between these segments is generally called "recoding". Strangly DQ4 is used for boot pin (held low during startup, this will force the ECU in boot mode).



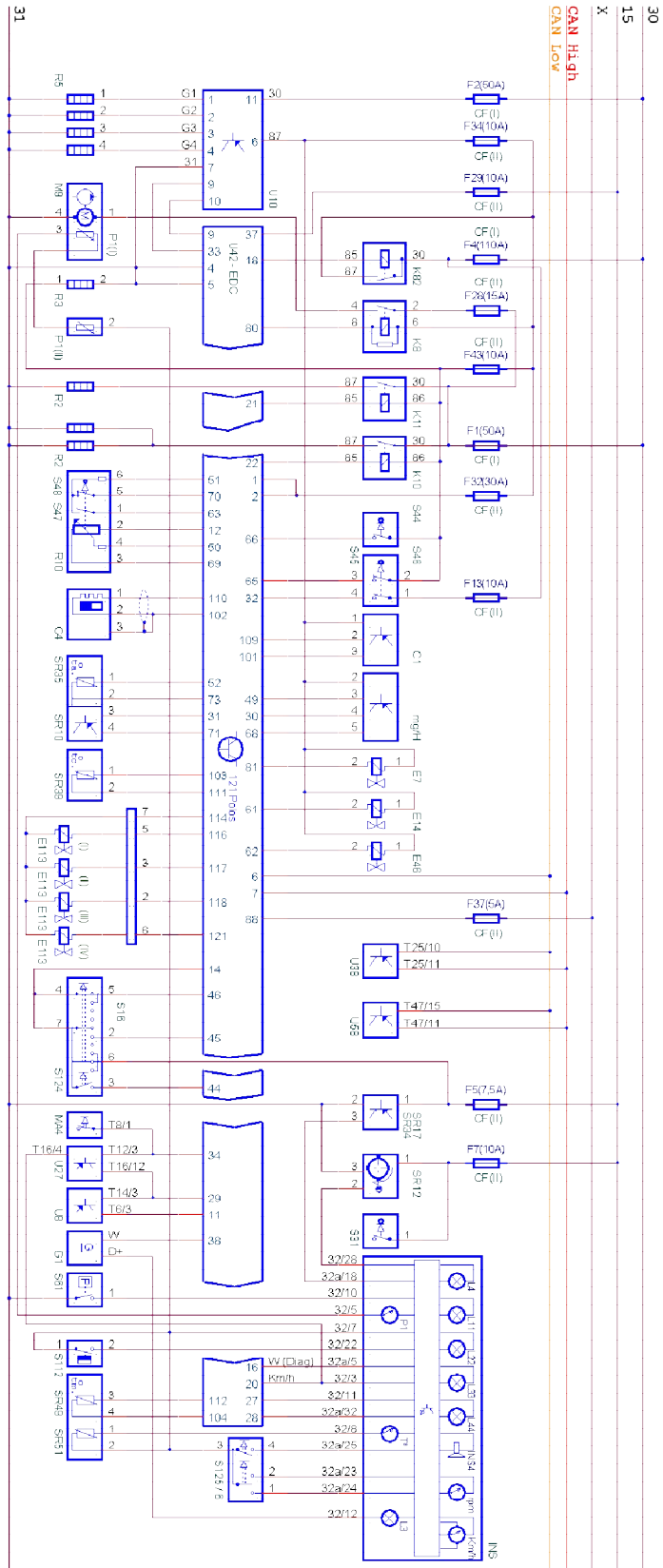
## ECU Pinout



**Ground**  
**+12V**  
**K-line**

**4 and 5**  
**2 and 37**  
**16**

Pin number	Color	Description
1	Red/violet	+12V (supply voltage)
2	Red/violet	+12V (supply voltage)
4	Brown/red	Ground
5	Brown/red	Ground
6	Orange/brown	CAN-L
7	Orange/black	CAN-H
16	Orange/black	K-line diagnostics + flashing (KWP1281 + KWP2000)
37		Ignition switch (switch +12V)

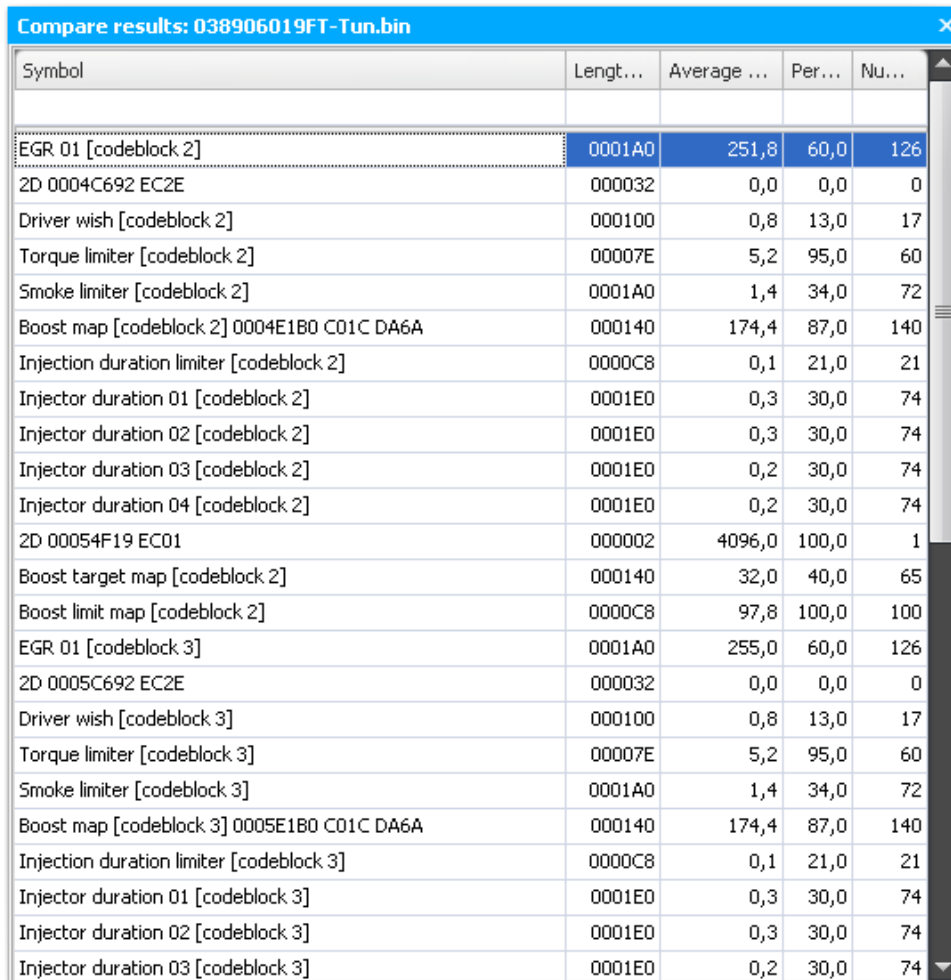




## Software

Once we download the data from the ECU with a MPPS, Galletto 1260, BDM or some other means to do a full read we can load the binary file into EDC15P Suite (see <http://trionic.mobix.eu>)

We can see the most important maps being automatically detected and we can change them to our likings. Be careful though, you need to know what you are doing. The software can generate a differences list between two files as well and if we compare a stock file to a tuned file we can see only a few maps get edited normally.



The screenshot shows a window titled "Compare results: 038906019FT-Tun.bin". It contains a table with the following columns: Symbol, Length, Average, Percentage, and Number of points. The table lists various engine maps and their values for two different states (codeblock 2 and codeblock 3).

Symbol	Length	Average	Percentage	Number of points
EGR 01 [codeblock 2]	0001A0	251,8	60,0	126
2D 0004C692 EC2E	000032	0,0	0,0	0
Driver wish [codeblock 2]	000100	0,8	13,0	17
Torque limiter [codeblock 2]	00007E	5,2	95,0	60
Smoke limiter [codeblock 2]	0001A0	1,4	34,0	72
Boost map [codeblock 2] 0004E1B0 C01C DA6A	000140	174,4	87,0	140
Injection duration limiter [codeblock 2]	0000C8	0,1	21,0	21
Injector duration 01 [codeblock 2]	0001E0	0,3	30,0	74
Injector duration 02 [codeblock 2]	0001E0	0,3	30,0	74
Injector duration 03 [codeblock 2]	0001E0	0,2	30,0	74
Injector duration 04 [codeblock 2]	0001E0	0,2	30,0	74
2D 00054F19 EC01	000002	4096,0	100,0	1
Boost target map [codeblock 2]	000140	32,0	40,0	65
Boost limit map [codeblock 2]	0000C8	97,8	100,0	100
EGR 01 [codeblock 3]	0001A0	255,0	60,0	126
2D 0005C692 EC2E	000032	0,0	0,0	0
Driver wish [codeblock 3]	000100	0,8	13,0	17
Torque limiter [codeblock 3]	00007E	5,2	95,0	60
Smoke limiter [codeblock 3]	0001A0	1,4	34,0	72
Boost map [codeblock 3] 0005E1B0 C01C DA6A	000140	174,4	87,0	140
Injection duration limiter [codeblock 3]	0000C8	0,1	21,0	21
Injector duration 01 [codeblock 3]	0001E0	0,3	30,0	74
Injector duration 02 [codeblock 3]	0001E0	0,3	30,0	74
Injector duration 03 [codeblock 3]	0001E0	0,2	30,0	74

Figure 1: differences between stock and tuned maps

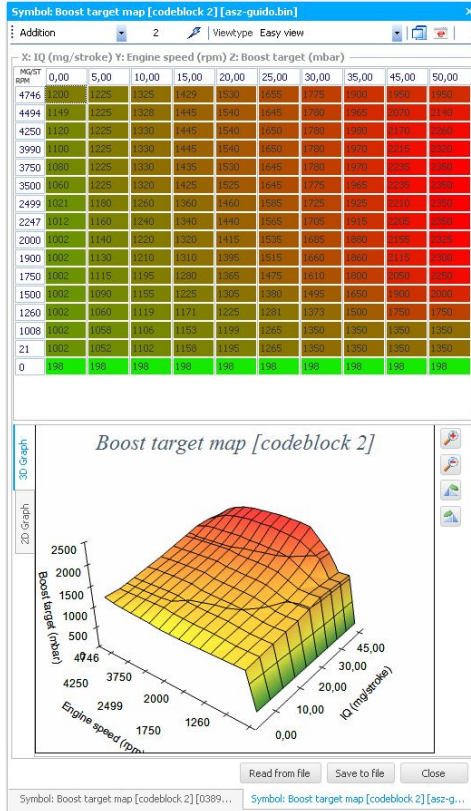


Figure 2: stock boost map

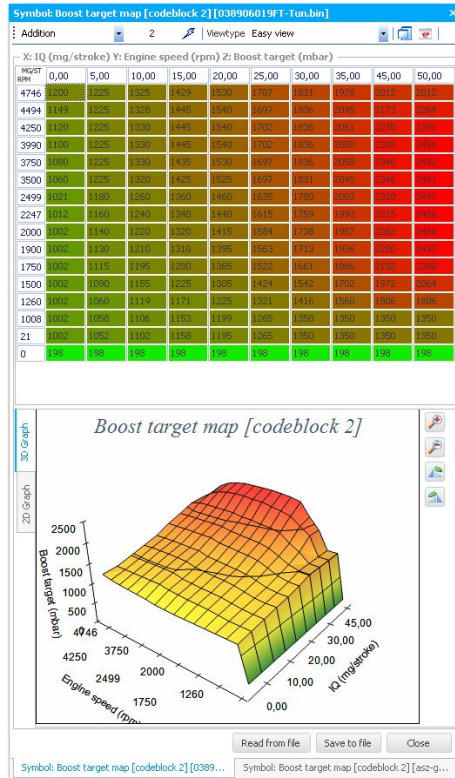


Figure 3: Tuned boost map

## Software information data

In the software, the identifiers are stored as well about HW revision, SW version, VAG partnumbers etc. This data is stored in ASCII in the binary file and looks something like this:

```
00010000h: 55 55 AA AA FE FF 3C 3C FD FF FF FF 3E 28 3F 53 ; UU^*pÿ<<ÿÿÿÿ>(?S
00010010h: FB FF FC FF 00 08 FF FF FF FF FF FF FF FF FF ; úÿÿÿ..ÿÿÿÿÿÿÿÿÿÿÿÿ
00010020h: F7 FF 41 41 31 31 32 32 37 37 30 30 33 33 46 46 ; ÷ÿAA1122770033FF
00010030h: 45 45 4C 48 6F 69 00 00 00 00 00 00 00 00 41 41 ; EELHoi.....AA
00010040h: EF FF AF 44 C0 FF 31 30 33 37 33 36 36 32 37 33 ; iÿ^DÀÿ1037366273
00010050h: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ; ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00010060h: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ; ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00010070h: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ; ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ

00053430h: F1 A0 E2 79 F2 A0 E3 79 F3 A0 E4 79 F4 A0 E5 79 ; ñ àÿò àÿó àÿô àÿ
00053440h: F5 A0 8B 3F 5F 47 30 33 38 39 30 36 30 31 39 46 ; ö <?_G038906019F
00053450h: 54 20 31 2C 39 6C 20 52 34 20 45 44 43 20 20 53 ; T 1,91 R4 EDC S
00053460h: 47 20 20 31 35 37 37 20 33 39 53 30 30 38 35 30 ; G 1577 39S00850
00053470h: 20 30 32 38 31 30 31 30 39 38 31 20 46 46 45 44 ; 0281010981 FFED
00053480h: 43 34 30 30 20 20 20 30 33 38 39 30 36 30 31 39 ; C400 038906019
00053490h: 46 54 20 30 34 2F 30 32 00 00 C8 00 14 00 78 05 ; FT 04/02..È...x.
000534a0h: E8 03 19 1A 4C 04 E8 03 85 86 78 05 E8 03 87 88 ; è...L.è...tx.è.+^
```

0281010981 is the hardware ID

1037366273 is the Bosch software ID

038906019FT is the VAG number

## Reading the code

To be able to understand the software better we'll need to dive into the world of assembler language. This is a sort of intermediate between understandable human language and the operation codes used by the microprocessor. Once we can read the assembler language (assembly for short) we can track all the things the microprocessor is told to do when the program is running. This is very valuable information because we don't have first hand information from either Bosch or VAG that can tell us in details what the ECU does.

We convert the binary file into assembly language we need to disassemble the file. We can do that by running a disassembler like IDAPro or a separate disassembler for the specific uC. This disassembler can be found here on the website.

Disassembler C167 <http://trionic.mobixs.eu/EDC15P/C167d.rar>

Once we disassemble the binary file we have an file containing the assembly listing in which we can start to explore and understand the internal workings of EDC15P.



The first two marked bytes (0x2E 0xEC) tell us it might be the start of an axis (the software has a list of known ids for this) It then validate the second pair of bytes (0x10 0x00). If this value looks like a valid length it starts to read data from that point on (0x10 byte pairs). Next it evaluates whether there is a second axis after the first one. The ID that is read is 0x36 0xC0 in this case and the length of the second axis is 0x0A 0x00. It now knows that this map is 16 x 10 values in dimensions, it knows the values for both axis (the data after the length indicators) and it knows the starting address and length of the map data. This procedure is done for all addresses found in the collection and the software stores the validated maps in a new collection.

### **Labelling the maps**

This is possibly the part that has the most assumptions in it. Most maps are named by looking at the dimensions and their axis IDs. In some files though, there are multiple maps with the same properties. In that case, the software also looks at (for example) the axis values or the map structure.

## Tuning EDC15P

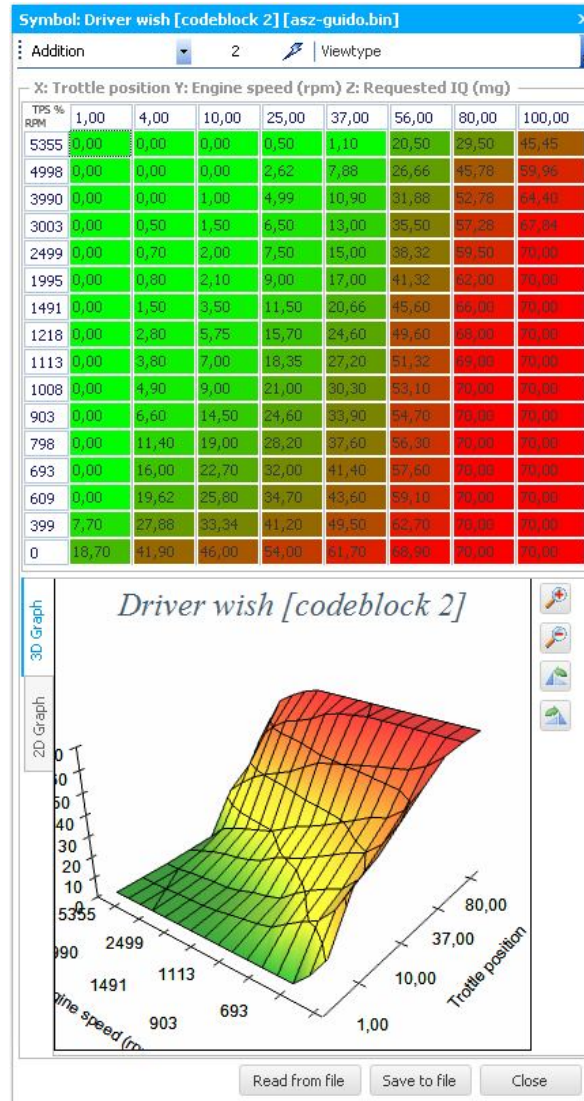
This chapter will describe the basic mapping you will need for a simple stage 1. Complicated mappings for altered hardware (airmass sensors, mapsensors, turbo's, bigger injectors etc) are not described here. It will also give a good overview on what the EDC15P Suite has to offer.

The maps needed for a simple stage 1 are;

- ✓ Driver wish map
- ✓ Torque limiter
- ✓ Smoke limiter
- ✓ Injection duration map
- ✓ EGR map (optionally)
- ✓ Boost target map
- ✓ N75 duty cycle map (e.g. wastegate/VG control)
- ✓ Boost limit map
- ✓ Single value boost limiter

## Driver wish map

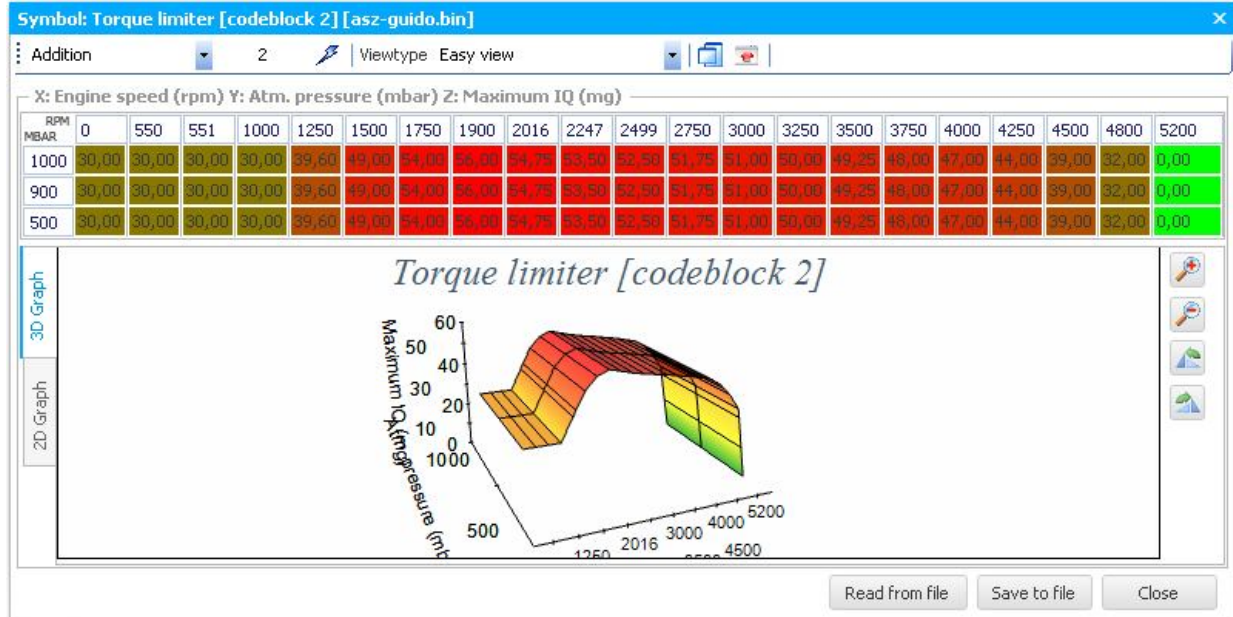
The driver wish map determines not only the maximum torque/power output of the car but also the driveability of it. It determines the amount of requested fuel for the percentage of accelerator pedal depression and engine speed (e.g. for every throttle position % and engine speed there is a value that tells the amount of requested fuel for that specific point).



As you can see, the maximum (for throttle) request amount of fuel is 70 mg/stroke in this file.

## Torque limiter

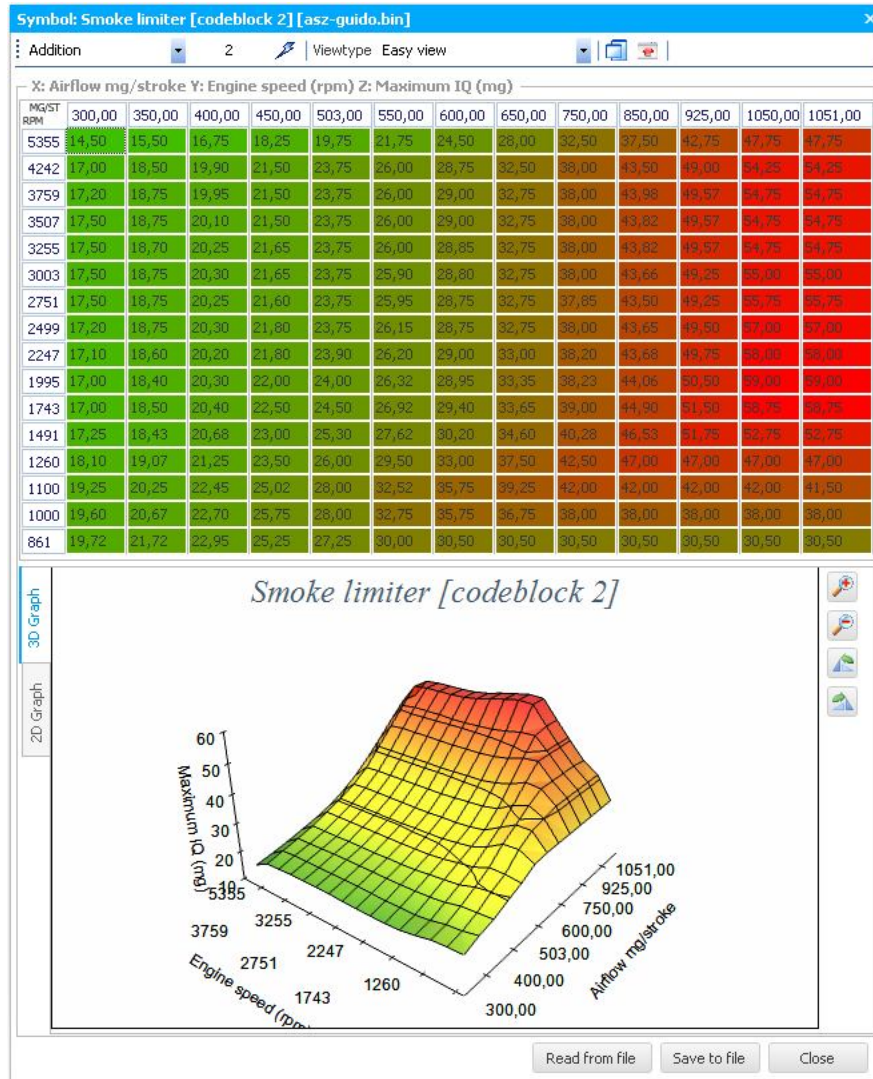
To prevent the transmission and drivetrain components to suffer too much from low end torque the output power is restricted in low engine speeds. It also lets you limit torque when atmospheric pressure is lower than average (e.g. high in the mountains). This is done to prevent the turbo from overrevving.





## Smoke limiter

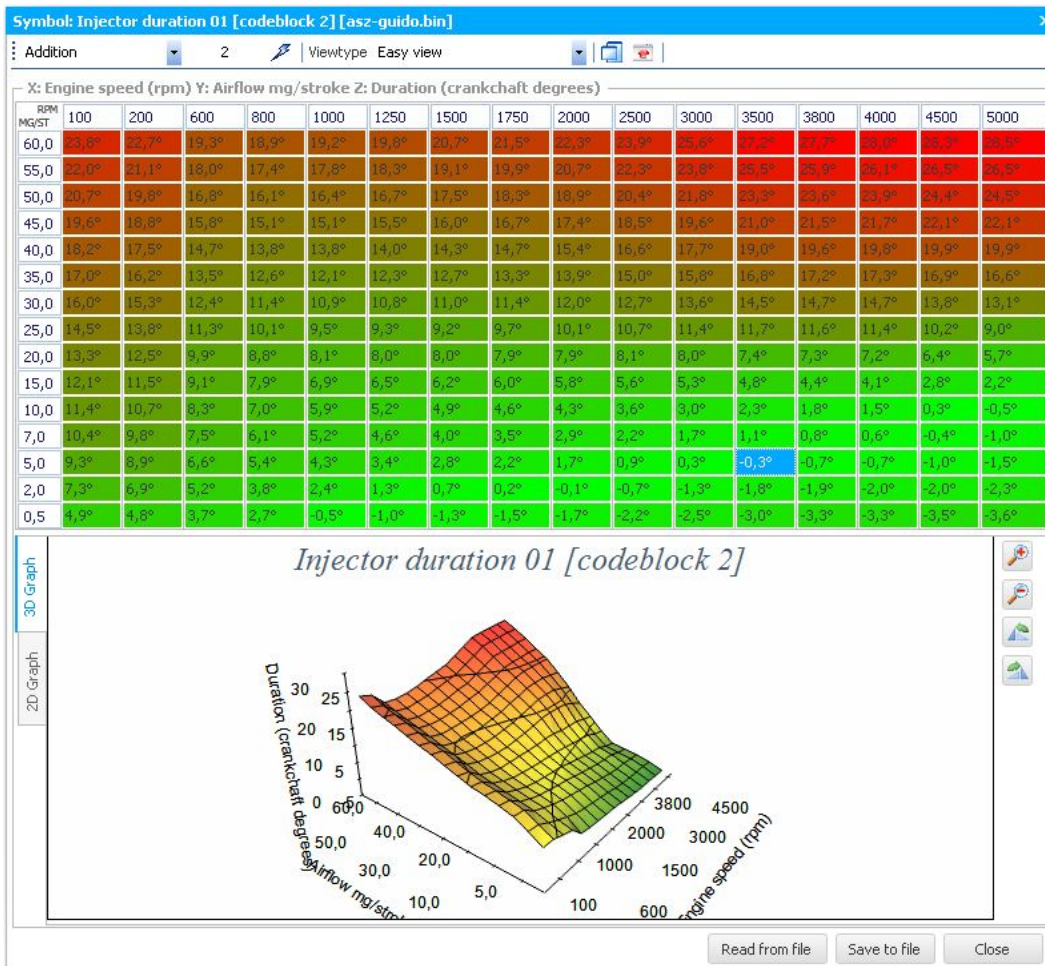
The smoke limiter is there to prevent excessive smoke to appear when the user depresses the accelerator pedal. Smoke is generated when the air to fuel ratio in a diesel engine are lower than 1:17. The smoke limiter tells the ECU not to inject more than the calibrated amount of fuel for any given amount of air entering the engine (airmass sensor data). If the driver wish map indicates (requests) 70 mg of air but there is only 1050 mg of air entering the engine the smoke limiter would limit the injected quantity to 55 mg of fuel @2750 rpm.



## Injection duration map

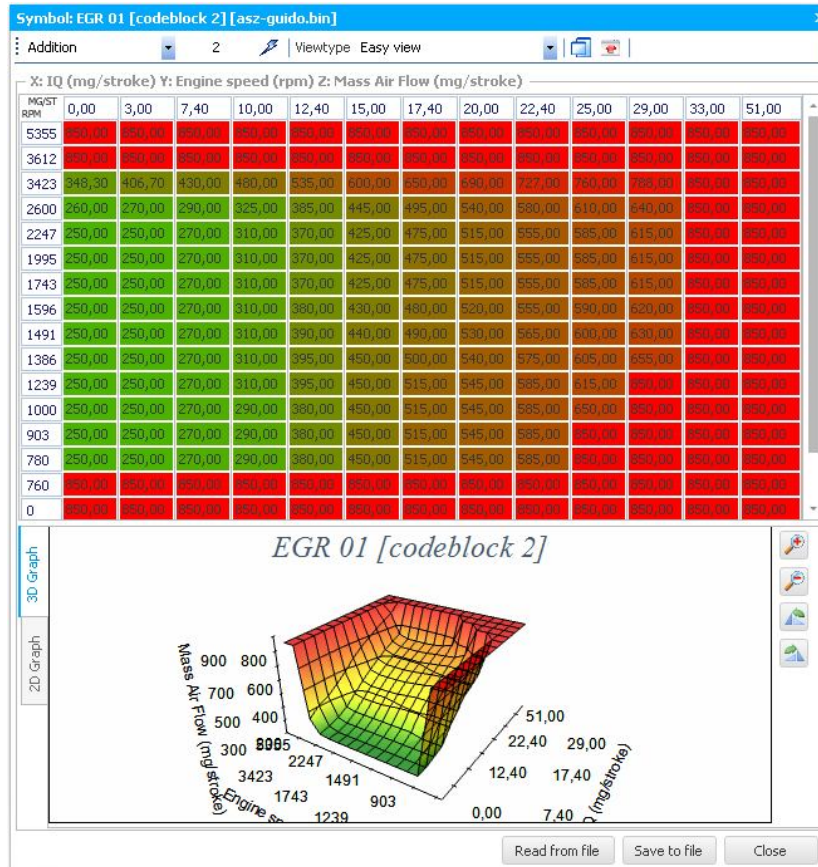
The injected amount of fuel determines the amount of power produced (upto a certain limit anyway). We can roughly estimate the amount of torque to be  $1.5 * \text{\#cylinders} * \text{IQ}$  (Injected Quantity). Injecting 60 mg/stroke in a 4 cylinder engine would then result in roughly  $1.5 * 4 * 60 = 360$  Nm of torque. Calculating power from torque can be done with this formula:  $\text{Power (hp)} = (\text{Torque (Nm)} * \text{rpm}) / 7121$ . Injecting the same 60 mg/stroke at 4000 rpm results in  $360 * 4000 / 7121 = 202$  hp.

This map tells the ECU how many crankshaft degrees it takes at a given engine speed to inject the requested amount of fuel.



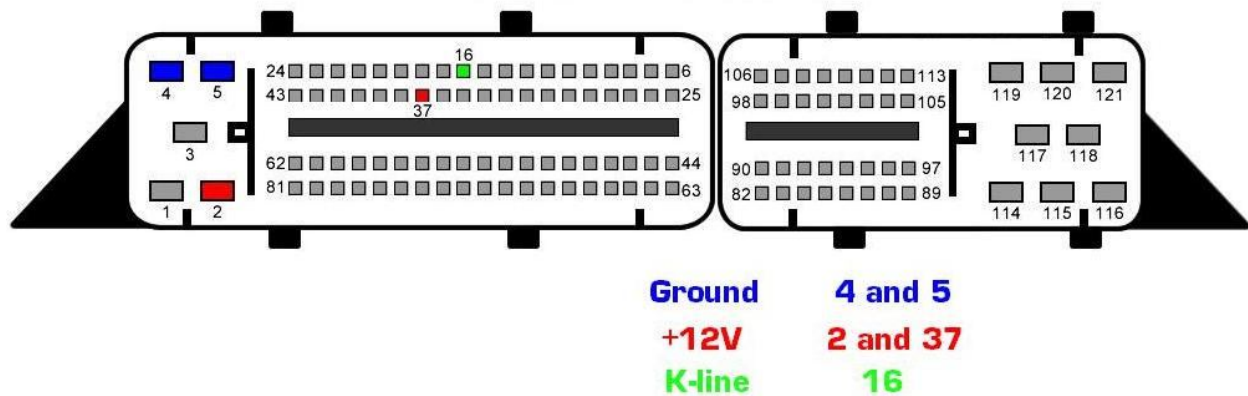
## EGR map

The EGR (Exhaust Gas Recirculation) map determines how much fresh air is allowed to enter the engine for a given injection quantity and engine speed whenever the EGR function is active. The remainder of the amount of air is fed back into the engine by the EGR system. Since this is hot, contaminated air, we don't want this in our engine from a performance point of view.

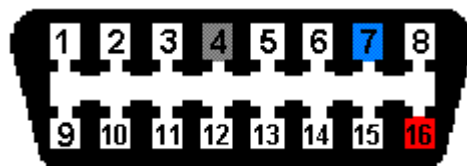


## Communication with the ECU

### Connection diagram



*ECU connector: This is looking at the connector on the ECU*



*This is the socket-part, connector part is mirrored*

There are three methods of communication that can be used with a EDC15P ECU.

- KWP1281
- KWP2000
- Boot mode communication

### KWP1281

To activate KWP1281 communication we need to connect a K-line interface to the ECU on pin 16 and after the 5 baud wake-up procedure communication can commence at 9600 baud.

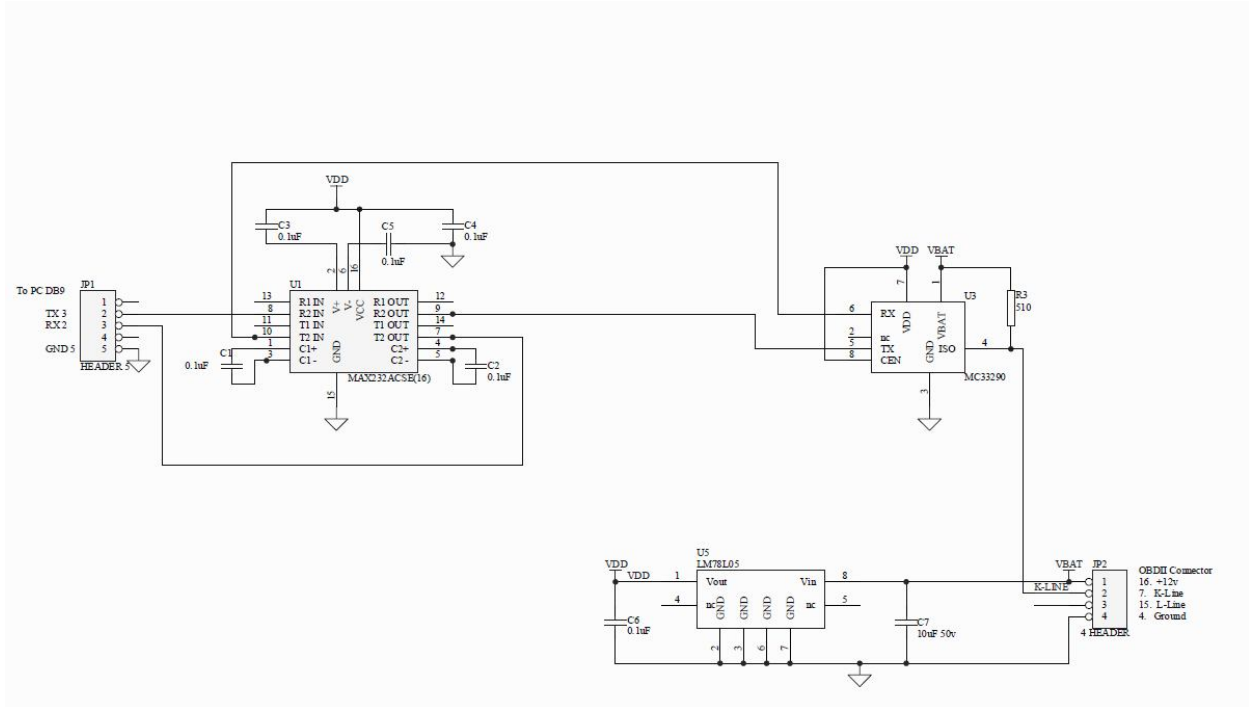
### Wakeup procedure for normal mode

To be able to communicate in normal mode, the ECU needs to be aware of the fact that there is a diagnostics device connected to pin 16 on the K-line. To let the ECU know we need to send a 0x01 byte to the port at 5 baud (!). After a correct wakeup byte on the 16 pin we will receive a response from the ECU at 9600 baud. This response will be 0x55 0x01 0x8A in which the 0x55 is the acknowledge and the 0x01 and 0x8A are the keywords used to communicate with the ECU. After reception of this sequence we need to send an acknowledge message to the ECU which is the inverted last keyword which will be 0x75.

# Appendix I: Building a high speed K-line interface

The following information is sourced from skpang.co.uk.

## Schematic



## Parts information

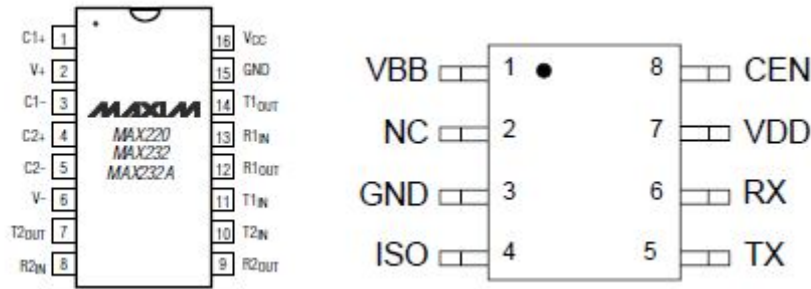


Figure 4: MAX232 and MC33290