Trionic 7 Documentation

Dilemma, typeset in LaTeXby Jonathan Jogenfors

September 12, 2015

Chapter 1

Introduction

This document is intended for Saab fanatics and engineers who want to start understanding the Saab Trionic 7 motor management system. It will give as much information as possible about the technical part of the system. The only limitation will be the knowledge of the author. In short the content of this document will enable you to understand Trionic 7 better and give you hands-on information about altering the maps it uses. Prerequisites are minor electronics and computer knowledge and of course some understanding of how a turbo charged engine works. Throughout the document the T7Suite software will be referenced. This software will enable you to really “get into” the Trionic. The T7Suite software can be downloaded from the T7Suite website.

1.1 Acknowledgements

The author would like to thank everyone on ecuproject for their help on getting all this information together. Special thanks go out to General Failure, J.K. Nilsson, Hook, Hma, Vigge, Mackan, Sandy Rus, JKB, L4staero and Steve Hayes.

Contents

Chapter 2

Hardware

The T7 is build around a Motorola MC68332 (CPU32) microcontroller. This is a 32 bit controller that handles the entire motor management including fuel injection, ignition timing and boost pressure control. The processor has a vast 4Mb (512 Kbyte) flash memory for fetching program code and maps. This flash memory consisting of a AM29F400BT-90SI (AMD) holds the program memory. There is a coprocessor from Philips, a P83C592FHA/019. This is a 8-bit 8051-based microcontroller with a CAN (Controller Area Network) module. As the CAN physical line driver there is an Intel AN82527 (same family as used in Trionic 5). RAM memory is done by two 32 Kbit SRAM chips (U62H256S1K). There is also a special component that would appear to be a barometric pressure sensor.

2.1 Integrated Circuit List

The table below lists almost all IC's on the board. This is just to give you an idea on what to expect.

2.2 Block Schematic Diagram

2.3 DI Cassette

The DI cartridge has a trigger input for firing the four individual sparkplugs. These are triggered by signals from the ECU on pin 9, 10, 11 and 12 which are generated in the power driver IC CA3236 on the Trionic PCB (topside, 16 pin DIL housing). Internally these four pins are connected as shown in the table and the image below.

Partnumber	Function	Usage	# on board
TC55257DFI-85L	SRAM	(working memory)	1
16233970	Microcontroller	Main 32-bit CPU	1
PC83C592	Microcontroller with CAN contr.	8-bit coprocessor	1
AM29F400	Flash memory	4 Mbit	1
51862	DA converter		1
AN82527	CAN controller CAN line driver	1	
16238669 0H11	Pressure sensor?		1

Table 2.1

Figure 2.1

DI cartridge pin	ECU pinnumber	CA3236 pinnumber	Description
2	7	1 (OUT A)	Trigger cylinder 1
3	8	3 (OUT B)	Trigger cylinder 2
4	67	6 (OUT C)	Trigger cylinder 3
5	68	8 (OUT D)	Trigger cylinder 4

Table 2.2

Chapter 3

Flashing

The Trionic 7 ECU binary images uses several checksums to verify integrity. Most of them have been easy to figure out, but one of them is so complicated, that it seems to been done to deter map changing. There are still some unknowns, that would be nice to figure out. For example, I've discovered some binaries don't seem to have all four checksums. And of course there could be more checksums that have gone unnoticed. Two of the checksums are in the end of the binary and the other two are scattered in the code. The latter ones can be found by using pattern searching. Again the calculations are pretty simple, and even the harder checksum is easy to implement. Big thanks to solving these things goes to Tomi and General Failure.

3.1 Checksum lexicon

First of all, there are four different checksums. They have been given names by Tomi: FB checksum, F2 checksum, Misc checksum and Area 70000 checksum.

3.2 F2 and FB checksums

The first two checksums, FB and F2, can be found at the end of the binary. This end area has been called the file header (footer would be more logical). See also chapter Trionic 7 file header. The F2 checksum is not present in all binaries, so be aware of this. Finding the two other checksums is more of a challenge.

Add reference

3.3 Misc. checksum

The Misc checksum resides inline within the code. It is usually found in the area of 0x02000...0x05000. The checksum address can be found by pattern searching the bin file using a set of hex values along with mask bits. If a mask bit is not set, the corresponding hex value does not have to match. Here are the hex values, and the masks. Pattern

```
0x48,0xE7,0x00,0x3C,0x24,0x7C,0x00,0xF0,0x00,0x00,0x26,0x7C,0x00,0x00,0x00,0x00,0x28,0x7C,0x
```

Mask

```
1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1
```

So, we are searching the binary for a string of bytes beginning with 0x48, 0xE7, 0x00, 0x3C... Then we mask out the bytes that change from binary to binary. When we've located this pattern, we know where to start. Now we start primitively disassembling the code. we search for byte patterns [0x48,0x6D], [0x48,0x78], [0x48,0x79], [0x2A,0x7C] and [0xB0,0xB9]. The three first patterns reveal addresses and lengths of checksum areas. There are 15 checksums areas from which the Misc checksum is calculated. The [0x2A,0x7C] pattern gives a base address for the [0x48,0x6D] addresses. Bear with me. These [0x2A,0x7C] addresses are summed with the base address to make the actual address. This way the address is only 2 bytes long. On the [0x48,0x79] addresses it's 4 bytes long without any base address. And the [0x48,0x78] pattern gives 2 bytes which correspond with the length of the checksum area. Finally the [0xB0,0xB9] pattern is followed by 4 bytes to the address of the Misc checksum.

3.4 Area 70000 checksum

This checksum refers to an area in the region of 0x70000. Like the Misc checksum, there is no clean way of finding out the length of the area along with the checksum address. Using pattern searching with this also has results. There are binaries that are incompatible with this approach, so this requires some fixing in the future. Pattern

```
0x20,0x3c,0x00,0x00,0x11,0x52,0x2F,0x00,0x20,0x3C,0x00,0x00,0x09,0xD0,0x2F,0x00,0x20,0x3C,0
```

Mask

```
1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1
```

After finding that pattern, the masked addresses are summed together. In the pattern, the original addresses are 0x00001152, 0x000009D0 and 0x000000CC. Summing these gives the Area 70000 length of 0x1BEE. At the same time this is the address where to find the Area 70000 checksum.

3.5 How to calculate a checksum

The FB checksum shares the same calculation method as Misc and Area 70000 checksums. It's simply a sum of the bytes from the checksum area. Four bytes are made into a 32 bit value and summed with the next 32-bit value. This goes on until there are fewer than 4 bytes left. The last 1...3 bytes are then individually summed together with the checksum.

3.6 Misc checksum

The Misc checksum is a sum of the individual checksums calculated from 15 areas. The checksum calculation used is the same as with the FB checksum.

3.7 Area 70000 checksum

Once you have found the length of the Area 70000, you can calculate the checksum by using the function described in section "FB checksum". The start address is 0x70000. Notice that your binary might not have this area present.

Chapter 4

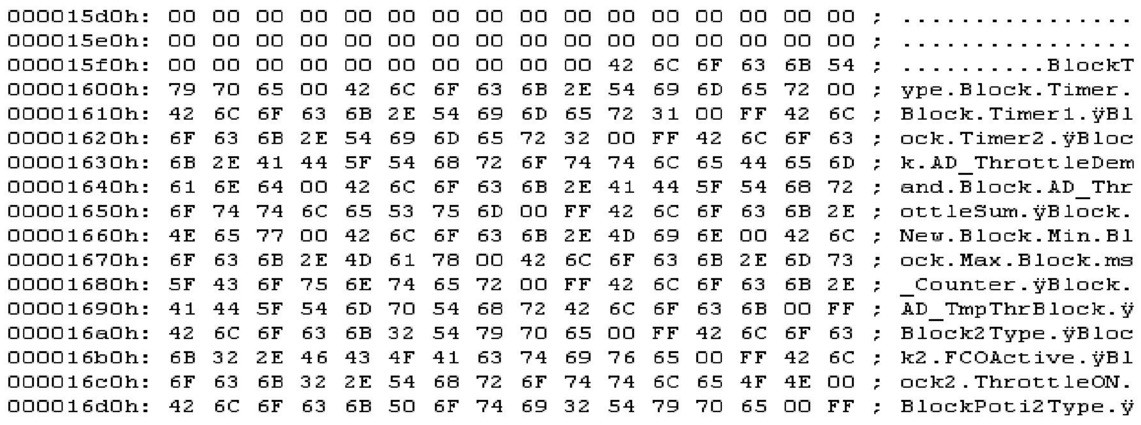
Firmware

Once you are done with dumping the flash contents and you want to do more than only alter variables and maps you can start analyzing the binary. This is a difficult task because there are a lot of different firmware versions, stock ones – maybe different per MY and tuned ones that differ for every manufacturer and stage. In every case the code can be disassembled using a 6833x disassembler like the one in IDAPro. There are scripts available to automatically disassemble the code and make it more readable by replacing addresses by variable names that are extracted from the symboltable inside the binary.

Chapter 5

Symbol table

Each T7 firmware file contains a symbol table describing data structures in the program. The major problem is that from some point in time SAAB started to compress the symbol tables in the binary file. Probably just to save space in the flash memory but it has made tuning a little harder. We actually need these symbol names because they tell us what a certain memory location means. For unpacked binaries these symbols can be extracted together with their corresponding memory addresses (ROM and RAM).



```
000015d0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
000015e0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
000015f0h: 00 00 00 00 00 00 00 00 00 00 42 6C 6F 63 6B 54 ; ..........BlockT
00001600h: 79 70 65 00 42 6C 6F 63 6B 2E 54 69 6D 65 72 00 ; ype.Block.Timer.
00001610h: 42 6C 6F 63 6B 2E 54 69 6D 65 72 31 00 FF 42 6C ; Block.Timer1.ÿBl
00001620h: 6F 63 6B 2E 54 69 6D 65 72 32 00 FF 42 6C 6F 63 ; ock.Timer2.ÿBloc
00001630h: 6B 2E 41 44 5F 54 68 72 6F 74 74 6C 65 44 65 6D ; k.AD_ThrottleDem
00001640h: 61 6E 64 00 42 6C 6F 63 6B 2E 41 44 5F 54 68 72 ; and.Block.AD_Thr
00001650h: 6F 74 74 6C 65 53 75 6D 00 FF 42 6C 6F 63 6B 2E ; ottleSum.ÿBlock.
00001660h: 4E 65 77 00 42 6C 6F 63 6B 2E 4D 69 6E 00 42 6C ; New.Block.Min.Bl
00001670h: 6F 63 6B 2E 4D 61 78 00 42 6C 6F 63 6B 2E 6D 73 ; ock.Max.Block.ms
00001680h: 5F 43 6F 75 6E 74 65 72 00 FF 42 6C 6F 63 6B 2E ; _Counter.ÿBlock.
00001690h: 41 44 5F 54 6D 70 54 68 72 42 6C 6F 63 6B 00 FF ; AD_TmpThrBlock.ÿ
000016a0h: 42 6C 6F 63 6B 32 54 79 70 65 00 FF 42 6C 6F 63 ; Block2Type.ÿBloc
000016b0h: 6B 32 2E 46 43 4F 41 63 74 69 76 65 00 FF 42 6C ; k2.FCOActive.ÿBl
000016c0h: 6F 63 6B 32 2E 54 68 72 6F 74 74 6C 65 4F 4E 00 ; ock2.ThrottleON.
000016d0h: 42 6C 6F 63 6B 50 6F 74 69 32 54 79 70 65 00 FF ; BlockPoti2Type.ÿ
```

Figure 5.1

While examining the symbol table you can see that the separator is 0x00. In contrast to T5 where symbol and SRAM addresses reside in the same table, we now only find the symbol name. In another table in the binary we can find flash addresses and lengths in the same sequence as the symboltable.

5.1 Finding start of symbol table:

Search binary for string the first sequence of 15 zeros.

5.2 Finding start of address lookup table:

Search binary for 20 00 00 00 XX YY 00 F0 where XX YY is the index of the first symbol found in the symbollist.

To save you the time to lookup all addresses manually the T7Suite application will extract all symbol information in one run. Symbol name, flash address and length will be displayed all together.

This image (6) will give you an idea of what the symbol table should look like once it has been extracted. See appendix I for a complete list of known symbols.

Figure 5.2

When the user double clicks one of the symbols that has a flash address attached to it, T7Suite will display the corresponding symbol in a viewer. This viewer will display the data in table form was well as in graphical form.

Figure 5.3

Chapter 6

Maps

A lot of maps in the T7 are not only made up of a piece of raw data. It also includes x-axis and y-axis information. T7Suite will automatically display all known axis information when a map is opened. In Trionic 7 most symbols have an English name (Trionic 5 has lots of Swedish names) that explains lots about its function. Also, the symbols are categorized by name, which makes browsing the symbols much easier. All torque calibration symbols start with “TorqueCal.”. T7Suite groups all symbols by their respective category by default.

6.1 Fuel

Fuel calculation in Trionic 7 is based on the Airmass entering the engine. In rough steps this seems to be the calculation’s flow:

Basic calculation of fuel quantity per combustion The current air mass/combustion is divided by 14.7 and sent to box 2. The unit is now in mg fuel/combustion

Compensation In case of a cold engine, shortly after starting, rapid load changes, knocking or high loads, the current value is multiplied by a compensation factor

Closed loop The closed loop value is used as a multiplier. The value is then sent to box 4

Correction for purge Multiply by the value for purge adaptation. The value is sent to box 5

Multiplicative adaptation (long term fuel trim) The multiplicative adaptation value is used as a multiplier and the new value is sent to box 6

Additive adaptation The additive adaptation value is added and the new value is sent to box 7

Starting fuel quantity If the engine has not yet started, starting fuel is selected. The value is sent to box 8

Fuel quantity per combustion to be injected The fuel quantity per combustion is the amount of petrol to be supplied to the engine. The value is sent to box 9

Injector opening duration Converts the value to the time during which the injector must be open and the new value is sent to box 10

Injection twice per combustion Injection takes place twice per combustion until the camshaft position has been found. Injection duration is divided by two. The value is sent to box 11

Voltage dependant needle lift duration added (battery correction) Adds the injector time delay, which is voltage dependant. The value is sent to box 12

Fuel cut The value is sent to box 13 unless fuel cut is active

Activation of injector At a DETERMINED crank shaft angle, the microprocessor will control the transistor for the injector that is next in the firing order

The basic fuel quantity is calculated based on Airmass and Injector constant. This injector constant is called InjCorrCal.InjectorConstant. If the engine is not warmed up yet, an alternate fuel map is used called

Figure 6.1

BFuelCal.StartMap. If the engine has reached operating temperature the normal map “BfuelCal.Map” is used.

Figure 6.2

You will see the areas calibrated to be run in closed loop have a value of around 1.00. It can be .98-1.02 or so. Then you will notice the high load part of the maps ramp up to enrich the mixture. The trick is to set the closed loop part of the map first, the areas that are represented by values of 1.00.

You will need to switch closed loop off and drive around with a wideband in the tailpipe. What you want to achieve is an AFR of 14.7 for petrol while driving around under light loads where the value is 1.00. What T7 does is calculate the injection time to be say 5ms, if that is not correct it is multiplied by this map. Say you need to enter a value of 1.1 in the map to get correct AFR it will change injection time to 5.5ms (5ms*1.1=5.5ms)

The idea is to get closed loop area correct first. This will stop any negative adaption once the tune is finished and allowed to run in closed loop again. If the closed loop area of your fuel map is too rich it will negatively adapt over a long period of time. This will have the effect of leaning your AFR’s across the board.

Example: you make a tune, closed loop AFR's are fine (because the O2 is making it fine through feedback) but unknown to you its rich and short term fuel trim is driving negatively 13leaning off injection time by 13%.

You don't notice this and make some full power runs to check AFR its fine at say 12.5 AFR. After several weeks the multiplicative adaptation (Long term fuel trim) has absorbed some adaption and has earned a value of -13% this will now subtract 13% from whole fuel calculation including full power. All of a sudden your full power AFR has jumped up to 14.0 AFR, *engine failure happens very easily from here*.

Now back to fuel mapping. To set closed loop area of fuel map, monitor the AFR in this light load area and if its wrong after adjusting for large injectors, start by just adjusting the Injector constant up and down accordingly instead of altering the closed loop area of the BfuelCalMap. This will affect the whole map instead of one point. By doing it this way you can almost get AFR spot on in closed loop area just by a few goes at adjusting the injector constant.

It can be found in InjCorrCal.InjectorConst, the value represents the injectors flow in mg of fuel (not capacity or cc's as injectors are normally rated in). The injector constant is a calculation factor used by T7 to calculate injection time.

Once closed loop area is done the high load area can be mapped. If its too lean just increase the values in the relative column relating to what site of the map you are running in. Once your high load areas are done, activate closed loop again so you can see how it all runs. Monitor fuel adaptations and AFR etc.

One thing to note is how quickly it drops into open loop under full throttle. By going into open loop the o2 sensor is "masked" where the ECU listens to what its saying but ignores it, this allows afr's to go beyond 14.7 and injection correction is directly taken from the fuel map we just adjusted allowing much enrichment to cool charge etc.

To alter open loop enrichment you can change at what Airmass flow T7 switches to open loop in LambdaCal.MaxLoadNormTab. Also, open loop entry (so, leaving closed loop situation) has a delay attached to it. This way, short overruns of the maximum load will not immediately result in leaving closed loop. Stock bins often have this set to 2000 milliseconds which seems quite long. If you want to ECU to leave closed loop faster after overrunning the load limit, just decrease the time in LambdaCal.TimeOpenLoop.

Figure 6.3: Battery correction values for injector latency

1. Idling speed ignition timing With idle speed control active, the timing is adjusted to stabilize idle engine speed. The value is sent to box 3

2. Normal ignition timing When idle speed control is inactive, the ignition timing is read from a load and engine speed depending matrix. The value from the matrix is optimized for lowest fuel consumption (best engine torque) and sent to box 3

Figure 6.4: Water temperature correction

Figure 6.5: Fuel injection correction map for knock conditions

3. Selection of ignition timing One of the ignition timing calculation is selected depending on which function is active. The value is sent to box 6
4. Catalytic converter heating timing In order to heat up the catalytic converter as fast as possible after start, the ignition will be retarded. This is a compensation matrix that is added to the value in box 3. The matrix is dependent on load and engine speed
5. Engagement of catalytic converter heating timing The function is active when coolant temperature is above -10 degrees Celsius and below +64 degrees Celsius
6. Total The value from box 5 is added to the value of box 3
7. Compensation The ignition timing is corrected depending on engine coolant temperature and intake air temperature. The value is sent to box 6.
8. Knock control If knocking occurs, a timing retardation will be calculated. The value is sent to box 6
9. Total The compensation angle and knock retardation are totalled to give the current ignition timing. The value is sent to box 7
10. Selection of ignition timing Starting ignition timing is selected when the engine has not been started. The value is sent to box 9
11. Starting ignition timing Starting ignition timing is selected when the engine has not yet been started. The value is sent to box 9
12. Activate relevant trigger At the calculated crankshaft angle, the microprocessor controls the transistor for the trigger that is next in firing order

Figure 6.6: ignition chart

6.1.1 DI Cassette

The ignition cassette is mounted on the valve cover on top of the spark plugs. The ignition cassette houses four ignition coils/transformers whose secondary coil is directly connected to the spark plugs. The ignition cassette is electrically supplied with battery voltage from the main relay (B+) and is grounded in an earth point. When the main relay is activated the battery voltage is transformed to 400 V DC which is stored in a capacitor. The 400 V voltage is connected to one of the poles of the primary coil in the four spark coils. Connected to the ignition cassette there are four triggering lines (from the Trionic ECU, pin 9 (cyl. 1), pin 10 (cyl. 2), pin 11 (cyl. 3) and pin 12 (cyl. 4)). When the ECU is grounding pin 9, the primary coil for the first cylinder is grounded (via the ignition cassettes B+ intake) and 400 V is transformed up to a maximum of 40 kV in the secondary coil for cyl. 1. The same procedure is used for controlling the ignition on the rest of the cylinders.

Figure 6.7: DI Cassette

6.1.2 Idle control

6.2 Torque

Trionic 7 is a torque/Airmass request system instead of a boost request system like Trionic 5 is. The basic procedure for the Airmass controller is like in the table below.

1. Driver request The control module reads pedal potentiometer 1 and converts the voltage to Airmass per combustion (mg/c). The value is sent to box 3
2. Cruise control request When cruise control is active, the air mass per combustion required to maintain the set speed is calculated. The value is sent to box 3

Figure 6.8: idlecontrol

Figure 6.9: Ignition is normally controlled by the main ignition matrix: IgnNormCal.Map

3. Select highest value The control module selects the highest of the two values (box 1 or box 2). The value is sent to box 5
4. Engine torque limitation The maximum permissible air mass per combustion varies depending on the engine type. During operation, the maximum permissible mg/c must also be limited to protect the engine, gearbox, brakes and turbo
5. Select lowest value The control module selects the lowest value and sends it to box 8
6. Compensation request When the AC compressor is on, and when the heated rear window or radiator fan is on, the mg/c required to compensate for the increased load is calculated. The value is sent to box 8
7. Other air request The control module calculates the mg/c required for idle speed control. The value is sent to box 8
8. Totalling values The control module totals all the values. The total is sent to box 9
9. Total requested mg/c
10. Total Airmass request
11. Throttle control The requested mg/c is converted to requested voltage for throttle position sensor 1. The charge air pressure and intake air temp are used to correct this conversion. The throttle motor rotates the throttle until the current voltage for throttle position sensor 1 corresponds with the requested voltage
12. Current mg/c The requested mg/c is also compared with the current mg/c (MAF reading). If needed the requested voltage for throttle position sensor 1 is finely adjusted

13. Turbo control If mg/c is too high for throttle alone the turbo control will take over. The excess is converted to a PWM which controls the charge air control valve. The absolute pressure sensor is used to correct the conversion

14. Current mg/c The requested mg/c is compared to current mg/c and the charge air control vale PWM is finely adjusted if required

6.2.1 Torque request

So, if the driver (or cruise control for that matter) pressed the accelerator pedal he actually requests a certain Airmass from the system. This value is fetched from the PedelMapCal.m_RequestMap shown below.

Figure 6.10: PedelMapCal.m_RequestMap

The table holds Airmass values for each position of the accelerator pedal and each rpm site. Trionic now looks up the estimated engine output (torque) based on Airmass and rpm. This is done through map "TorqueCal.M_NominalMap" as shown next.

Figure 6.11: TorqueCal.M_NominalMap

6.3 Second lambda sensor

In the years Trionic 7 was shipped on cars, several things changed in these cars setups. One of the major changes was the introduction of the second lambda (oxygen, O2) sensor that is placed after the catalyst to ensure the catalyst is working properly. If you want to run software from a double lambda sensor car in a single lambda car, you have to make some changes in the settings. This information is courtesy of L4staero.

Map	Value	Description
LambdaCal.ST_AdapEnable	0	Second lambda sensor disabled
LambdaCal.ST_AdapEnable	1	Second lambda sensor enabled

Table 6.1

Map	Value	Description
O2HeatPostCal.I_LowLim	0	
CatDiagCal.LoadLo	20	Second lambda sensor disabled
CatDiagCal.LoadHi	30	
O2HeatPostCal.I_LowLim	230	
CatDiagCal.LoadLo	140	Second lambda sensor enabled
CatDiagCal.LoadHi	425	

Table 6.2

6.3.1 Turning off second lambda sensor

You can use this procedure in cars having only one lambda sensor and in cars having two lambda sensors, but with a missing catalyst.

6.3.2 Alternative solution to turning off second lambda

Change low limit on O2heaterPostCal.I_LowLim to 0 mA to disable sensor heater error, and change CatDiagCal.LoadHi and LoadLo to values never seen normally, like 30 and 20.

6.4 Calibration of OBD2 and LEV EVAP systems

If we want to run a file that was developed for OBD2 or a LEV car in an earlier car we run into problems because the early car is missing a second catalyst, a tank pressure sensor and a purge canister behind the fuel tank. If you have an early B205E/L engine you simply couldn't run a later B205R software version in it because it would through CEL's for the missing hardware. We need to make changes to the file before we can run in on an earlier car (e.g. switch of the control of the new hardware).

OBDCal.OBD2Enabled= This is self explanatory, if car is OBD2 put value at 1 if it's not OBD2 put value at 0.

On this point in later bins(compressed) there is EOBDEnable which is always on in EC2000 EU files and LOBDEnable which is always on in EC2000 RW files. File type is shown in firmware information under engine type.

OBDCal.EnableOBD2Limit= As above but its a 4 byte value. If done in Hex value for a OBD2 car is 00000001 and for non-OBD2 car is 00000000. As shown in T7suite is 2 values. OBD2 cars top value is 1 and bottom value is 0. In non OBD2 car both values are 0.

OBDCal.evapEquipmentExist= If car is equipped with a canister at rear of tank and a tank pressure sensor value will be 1. If neither exist value should be set to 0.

6.4.1 Info on LEV (Low Emission Vehicle)

For example take a 2001 9-3 Aero (or SE as called in USA) equipped with a B205R, Saab's decision to take all "R" engines and clean them so to speak by developing new emission systems for them leaves them with some differences to their low level engine relatives. The term LEV(Low emission vehicle) in

Identifier	Length	Description
0x91	0x09	Ecuid.vehicleidnr
0x94	0x07	Ecuid.ecuhardwversnr
0x95	0x0C	Ecuid.ecusoftwnr
0x97	0x1E	Ecuid.ecusoftwversnr
0x9A	0x04	Ecuid.softwaredate
0x9C	0x04	variable name table crc (not really sure)
0x9B	0x04	Symboltable (packed table with symbol names)
0xF2	0x04	F2 checksum
0xFB	0x04	Romchecksum.piareachecksum
0xFC	0x04	Romchecksum.BottomOffFlash
0xFD	0x04	RomChecksumType
0xFE	0x04	Romchecksum.TopOffFlash
0xFA	0x05	Lastmodifiedby
0x92	0x0F	Ecuid.partnralphacode (IMMO)
0x93	0x07	Ecuid.ecuhardwnr
0xF8	0x02	?
0xF7	0x02	?
0xF6	0x02	?
0xF5	0x02	?
0x90	0x11	Ecuid.scaletable (VIN)
0x99	0x06	Ecuid.testerserialnr
0x98	0x0D	Ecuid.enginetype
0xF9	0x01	Romchecksum.Error

Table 6.3

this sense refers to Saab's decision to add a second catalytic converter and a tank pressure sensor and a large purge canister behind fuel tank, as well as adding a 2nd oxy to monitor condition of first cat.

6.5 Footer information

If we look at the footer in the binary (last page in hex viewer) we see a set of reversed strings. Each of these strings contains an identifier. These identifiers have a hardcoded meaning.

Figure 6.12: footer

Chapter 7

Tuning with T7

7.1 Tuning with T7Suite

To get the ECU to produce more engine output, several parameters (maps) have to altered. This chapter will give you a general idea on what to change – and why – for getting to an approximate stage II equivalent. The example is a 9-3 B205R.

7.1.1 AirCtrlCal.m_MaxAirTab

Airmass value from controller where area map has reached max-area and there is no point to increase the I-part. Resolution is 1 mg/c

Figure 7.1: AirCtrlCal.m_MaxAirTab