MASTERs 2014
LAB Manual for 18060 IVN
Interfacing with Vehicle Networks: Best Practices

Table of Contents

LAB 1:
Access OBD Data

Purpose:

Become familiar with the development tools: OBD development board and OBD simulator, and access vehicle data: vehicle speed, RPM, DTCs, and VIN.

Overview

In this lab, we will set up the OBD development board and the OBD Simulator, connect to them from the PC using a terminal emulator, and request and interpret vehicle parameters.

OBD Simulator

As the name suggests, the purpose of this device is to simulate the on-board diagnostic system of a vehicle. The ECUsim 2000 is capable of simulating all legislated OBD protocols, and supports a wide range of diagnostic services and parameters.

The unit, shown in Figure 1, has five knobs assigned to common PIDs, including vehicle speed and engine speed (RPM). The “Fault” button sets stored, pending, and permanent DTCs. The USB port can be used to monitor OBD traffic, and configure the simulator.

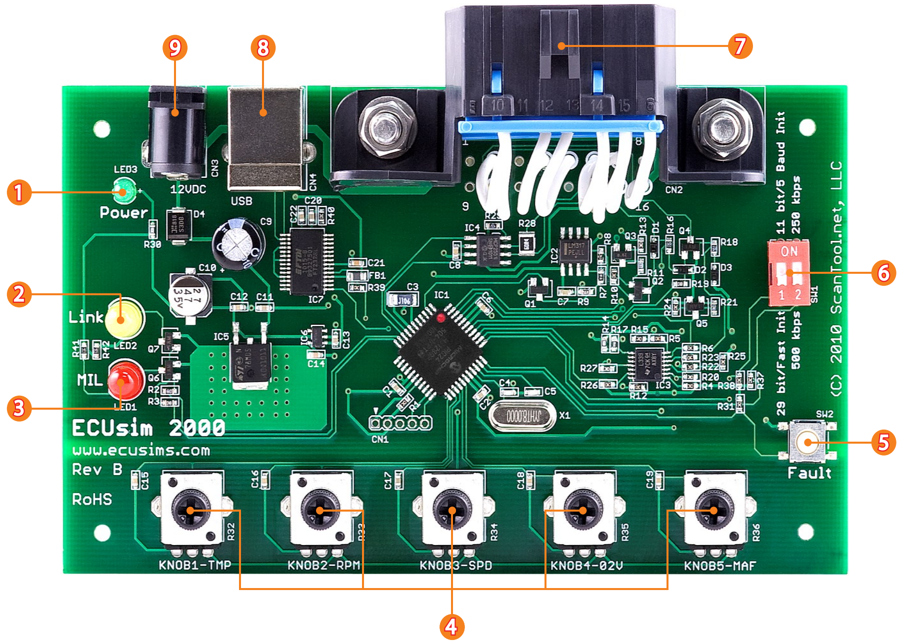
OBD Development Board

The OBD development board (Figure 2) is a fully functioning OBD to USB interface. It consists of the “interconnect” board and three modules:

- **OBD Interpreter Module:** a PIC24H-based intelligent OBD controller
- **OBD Transceiver Module:** provides level-shifting to/from OBD/TTL
- **Power Module:** provides load dump/reverse polarity protection, filtering out of spikes and dips, switches off peripherals in sleep, regulates voltage down to 5V and 3.3V

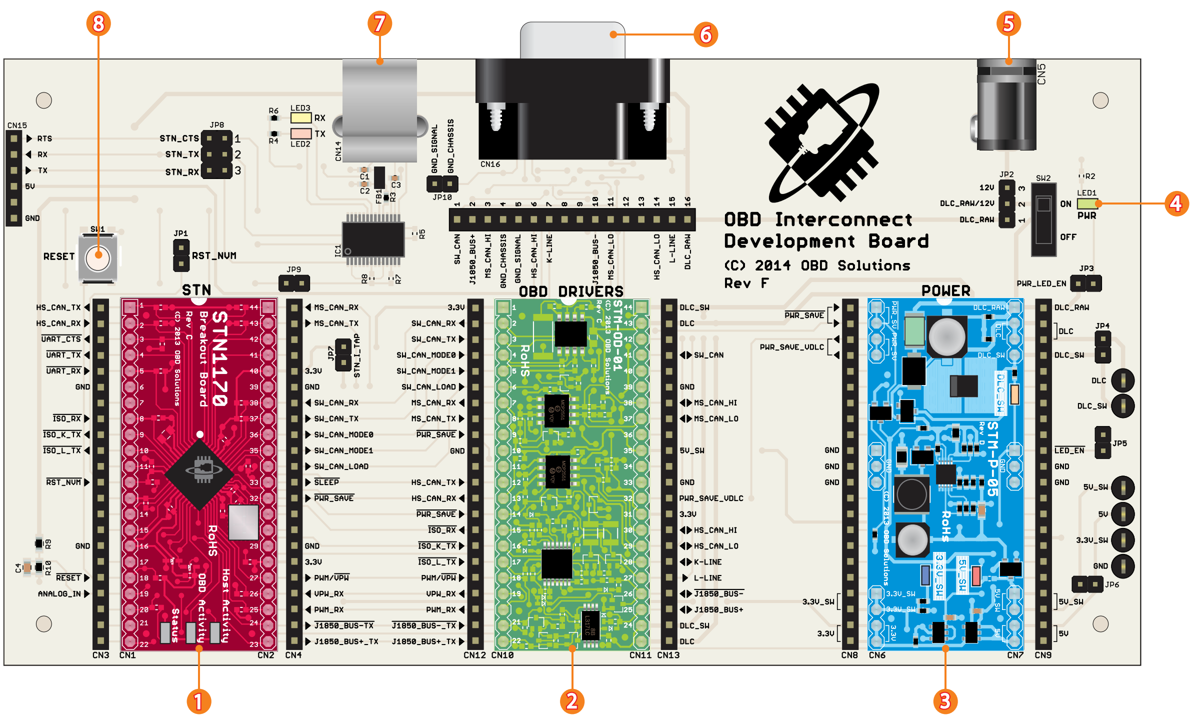
The board can be powered either via the power jack, or the OBD port. It features a number of jumpers and tap points, to facilitate experimentation during development and troubleshooting.

Figure 1: OBD Simulator



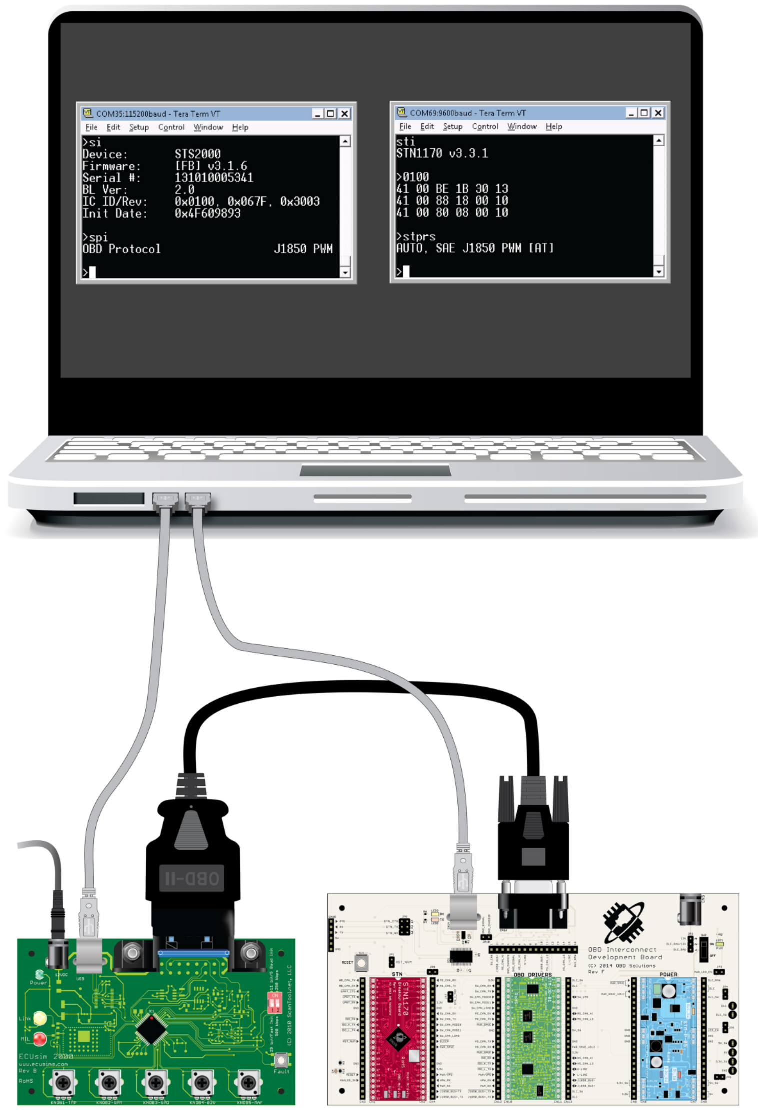
1 - “Power” LED
2 - “Link” LED
3 - “Malfunction Indicator Light” LED
4 - PID Control Knobs
5 - “Fault” button
6 - Configuration DIP switch
7 - OBD port
8 - USB port
9 - Power jack (12VDC)

Figure 2: OBD Development Board



1 - OBD interpreter module (STN1170)
2 - OBD transceiver module
3 - Power module
4 - “Power” LED
5 - Power jack (12VDC)
6 - OBD port

Figure 3: Connection Diagram



Step 1: Set up OBD simulator and Development Board

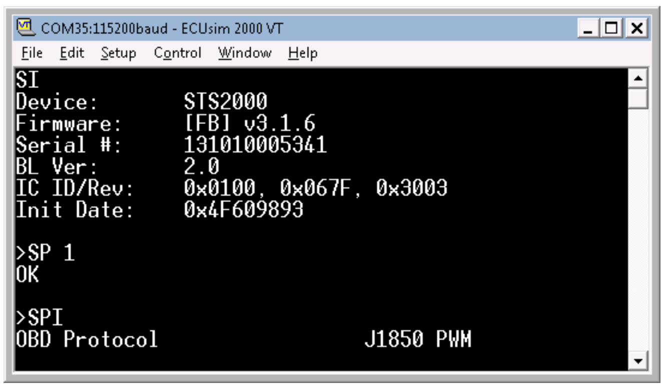
1. Plug the 12V end of the **power supply** into the power jack of the **OBD simulator.** *The "Power" LED on the OBD Simulator should light up.*
2. Connect the **development board** to the **OBD simulator,** using the OBD to DB15 cable. *The "Power" LED on the development board should light up.*
3. Launch **TeraTerm,** select the COM port, and set it up for 115200 kbps. This will be your **<u>"OBD Simulator"</u> terminal.**
4. Connect the **development board** to the **PC** using the USB cable.
5. Launch a **2nd TeraTerm** window, select the new COM port, and set it up for 9600 kbps. This will be your **<u>"Development Board"</u> terminal.** *Hit 'Enter' to test communication.*
6. Connect the **OBD simulator** to the **PC** using the 2nd USB cable.

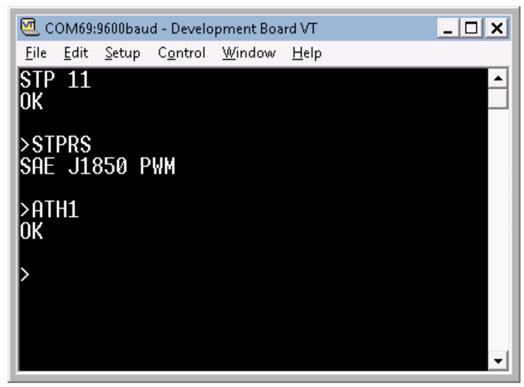
Step 2: Request and Interpret OBD Data

In this step, we will set both the simulator and the development board to the same OBD protocol, send OBD requests, and receive and interpret responses.

Type the following commands in their respective TeraTerm windows:

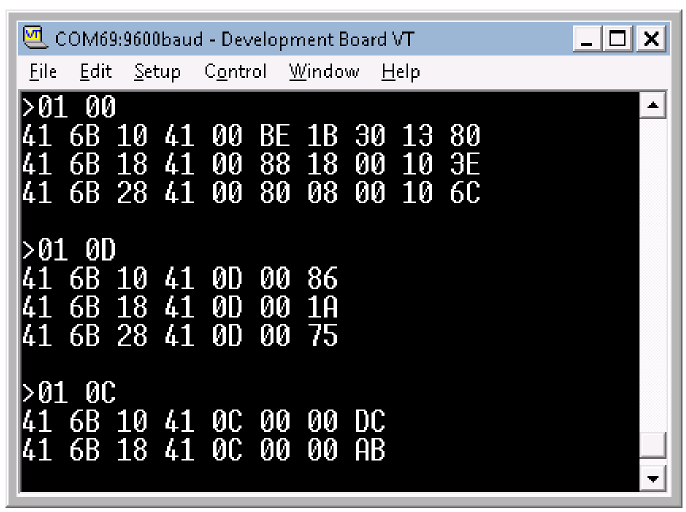
```
OBD Simulator:                          Development board:

SI          print device version        STP 11      protocol = J1850 PWM
SP 1        protocol = J1850 PWM        STPRS       print protocol
SPI         print protocol              ATH1        turn headers on
```





Now, you're ready to request OBD data. Enter the following commands in the "Development board" window:

```
01 00   supported PIDs
01 0D   vehicle speed
01 0C   RPM
```



– *How many ECUs responded to each request?*
– *What are their addresses?*

Turn the "SPD" and "RPM" knobs, and repeat the requests (you can hit 'Enter' to repeat the last command).

– *Which bytes are changing, in each case?*
– *What are the minimum and maximum values (in hex)?*

Turn the knobs roughly half a turn, and record the hex values for

Speed: _______________ RPM: __________________

Responses to the 0100 request give you three important pieces of information:

- How many ECUs on the network support legislated OBDII PIDs
- Address of each ECU
- Which PIDs are supported by each ECU

ECU count is equal to the number of responses you get to 0100. Address of the ECU is encoded in the header. Supported PIDs are bit-encoded in the data bytes, like this (from SAE J1979, Digital Annex):

Supported PID/OBDMID/ TID/INFOTYPE (Hex)	Scaling/Bit Number of Data Bytes = 4 Data A - D or B - E: Bit Evaluation PID/OBDMID/TID/INFOTYPE Supported (Hex)		
00	Data A bit 7 Data A bit 6 : Data D bit 0	01 02 : 20	0 = not supported 1 = supported

In our example, there were three responses, and looking at the third byte, we know the ECU addresses are 10, 18, and 28.

To learn how to determine which PIDs are supported by an ECU, let's look at the first response from our example:

```
41 6B 10 41 00 BE 1B 30 13 80
```

The first three bytes are the header, bytes 4 and 5 mean "this is a response to 01 00", the next four bytes are the data, and the last byte is the CRC (checkbyte).

Let's convert the hex bytes to binary, and count the bits to see which PIDs are supported:

PID	01	02	03	04	05	06	07	08
0xBE =	1	0	1	1	1	1	1	0

PID	09	0A	0B	0C	0D	0E	0F	10
0x1B =	0	0	0	1	1	0	1	1

PID	11	12	13	14	15	16	17	18
0x30 =	0	0	1	1	0	0	0	0

PID	19	1A	1B	1C	1D	1E	1F	20
0x13 =	0	0	0	1	0	0	1	1

Note that the last bit indicates that there are more supported PIDs, in the next range (PIDs 20-40). Send the 0120 request and repeat the process for the next 32 PIDs. If the last bit (PID 40) is supported, send 0140, and so on.

Use the following PID definition to decode the vehicle speed value you recorded earlier:

PID (hex)	Description	Data Byte	Min. Value	Max. Value	Scaling/Bit	External Test Equipment SI (Metric) / English Display
0D	Vehicle Speed Sensor	A	0 km/h	255 km/h	1 km/h per bit	VSS: xxx km/h (xxx mph)
	VSS shall display vehicle road speed. Vehicle speed may be derived from a vehicle speed sensor, calculated by the ECU using other speed sensors, or obtained from the vehicle serial data communication bus.					

In other words, to get vehicle speed (in km/h), simply convert the hex value to decimal. For example:

Raw coolant temperature value (hex): 6E
0x6E = 110
110 - 40 = 70
Coolant temperature is 70°C

Engine speed (RPM) definition:

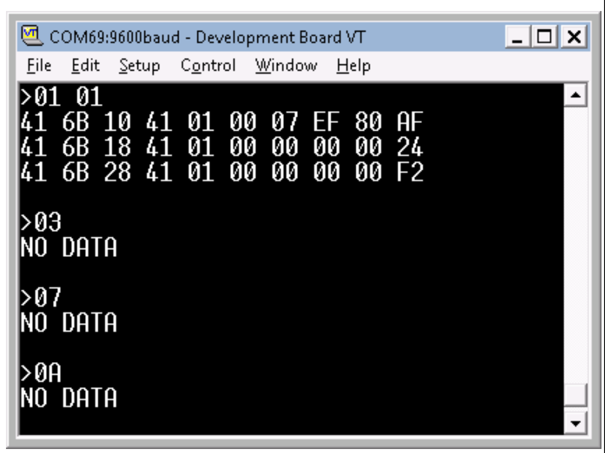
PID (hex)	Description	Data Byte	Min. Value	Max. Value	Scaling/Bit	External Test Equipment SI (Metric) / English Display
0C	Engine RPM	A, B	0 min^{-1}	16383.75 min^{-1}	1/4 rpm per bit	RPM: xxxxx min^{-1}
	Engine RPM shall display revolutions per minute of the engine crankshaft.					

In other words: convert the 16-bit hex value you recorded to decimal, and divide by 4 to get RPM. For example:

Raw RPM value (hex): 0EC4
0x0EC4 = 3780
3780 ÷ 4 = 945
RPM is 945 min^{-1}

Let's now request status of the MIL, DTC count, and DTCs:

```
01 01  MIL status
03     stored DTCs
07     pending DTCs
0A     permanent DTCs
```

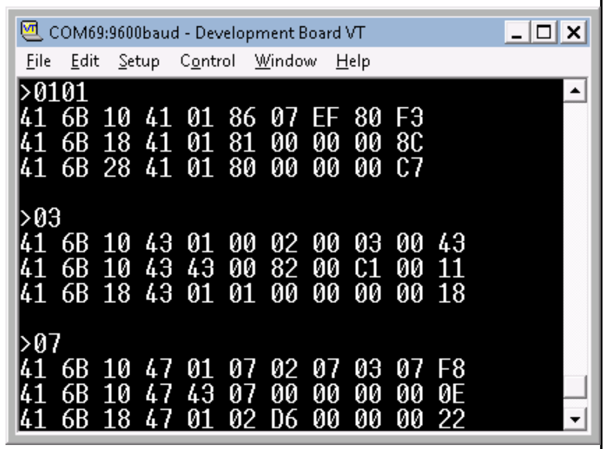


MIL status and DTC count are encoded in the first data byte. In our example, it's 00, meaning MIL is off and there are no DTCs.

Requests for stored, pending, and permanent DTCs return "NO DATA" — ECUs have no DTCs to report.

Press the “Fault” button on ECUsim, and repeat the requests:

```
01 01  MIL status
03     stored DTCs
07     pending DTCs
0A     permanent DTCs
```



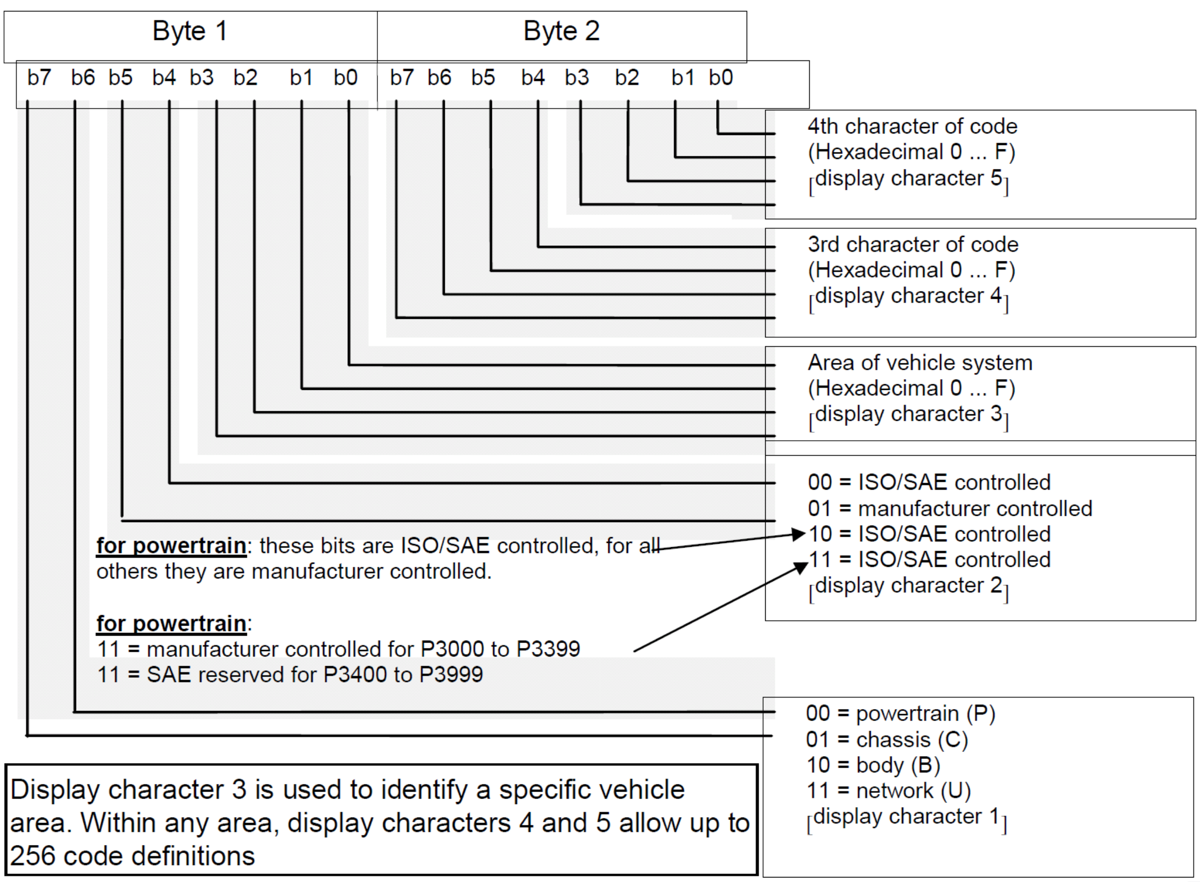
The first byte of the first response to 0101 is 0x86. The first bit is now 1, meaning the MIL is on. The DTC count is 6.

The next two responses are 81 and 80, meaning that ECUs number 18 and 28 have 1 and 0 DTCs, respectively.

Let’s now dissect the response to the request for stored DTCs (03):

```
41 6B 10 43 01 00 02 00 03 00 43
41 6B 10 43 43 00 82 00 C1 00 11
41 6B 18 43 01 01 00 00 00 00 18
```

ECU #10 responded with two messages, containing 6 DTCs.
ECU #18 responded with a single message, containing one DTC.



The SAE J2012 diagram shows how the DTCs are encoded. Notice how the last three characters of a DTC *do not need to be translated from ASCII.* Therefore, the only part that requires any effort, is decoding the letter-number combination (first nibble). Here's a handy chart to help us with this task:

	0	**1**	**2**	**3**
P	0	1	2	3
C	4	5	6	7
B	8	9	A	B
U	C	D	E	F

This is how we would decode the first, fourth, and sixth DTCs from our example:

01 00 = P0100
43 00 = C0300
C1 00 = U0100

Try decoding the rest of the DTCs, on your own.

As our final exercise of this lab, we will request and decode the VIN (0902). Besides the fields you are familiar with, the VIN response frames add a sequence number field. Also, the first frame starts with three fill bytes.

Send 09 02 to the ECUsim, and you will get the following response:

```
41 6B 10 49 02 01 00 00 00 31 2F
41 6B 10 49 02 02 47 31 4A 43 1D
41 6B 10 49 02 03 35 34 34 34 EC
41 6B 10 49 02 04 52 37 32 35 E9
41 6B 10 49 02 05 32 33 36 37 4C
```

Use the ASCII code chart, to decode the VIN:

ASCII Code Chart

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

VIN: ____

LAB 2: Sleep/Wakeup

Purpose:

To explore the various sleep/wakeup mechanisms.

Overview

In this lab, we will show the steps required to configure and activate sleep and wakeup triggers.

Automatic Sleep and Wakeup on UART (In-) activity

We will enter the following commands in the “Development Board” TeraTerm window:

STSLCS	*print sleep summary*
STSLUIT 10	*sleep after 10 seconds of inactivity*
STSLU on, on	*enable both sleep and wakeup triggers*
ATZ	*reboot to activate*
STSLCS	*check new trigger settings*
[wait 10 s]	
[hit spacebar]	
STSLU off, on	*disable sleep on UART in-activity*

Watch the LEDs on the Power Module, as you wait for the STN1170 to go to sleep. When they turn off, you can hit the spacebar (or any key) to wake up the device. When you do, the STN1170 will wake up and print the welcome prompt.

Type the last command, to disable the “sleep on UART inactivity” trigger.

COM69:9600baud - Development Board VT

File Edit Setup Control Window Help

```
>STSLCS
CTRL MODE:   NATIVE
PWR_CTRL:    LOW PWR = LOW
UART SLEEP:  OFF, 1200 s
UART WAKE:   ON,  0-30000 us
EXT INPUT:   LOW = SLEEP
EXT SLEEP:   OFF, LOW FOR 3000 ms
EXT WAKE:    ON,  HIGH FOR 2000 ms
VL SLEEP:    OFF, <13.00V FOR 600 s
VL WAKE:     OFF, >13.20V FOR 1 s
VCHG WAKE:   OFF, 0.20V IN 1000 ms

>STSLUIT 10
OK

>STSLU on, on
OK

>ATZ


ELM327 v1.3a

>STSLCS
CTRL MODE:   NATIVE
PWR_CTRL:    LOW PWR = LOW
UART SLEEP:  ON,  10 s
UART WAKE:   ON,  0-30000 us
EXT INPUT:   LOW = SLEEP
EXT SLEEP:   OFF, LOW FOR 3000 ms
EXT WAKE:    ON,  HIGH FOR 2000 ms
VL SLEEP:    OFF, <13.00V FOR 600 s
VL WAKE:     OFF, >13.20V FOR 1 s
VCHG WAKE:   OFF, 0.20V IN 1000 ms

>

ELM327 v1.3a

>STSLU off, on
OK

>
```

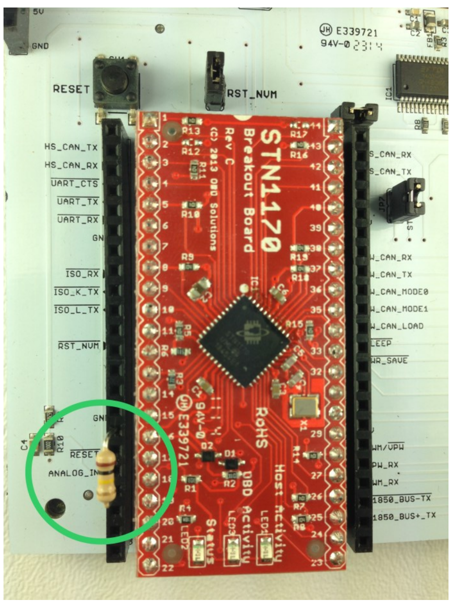
SLEEP Command and Wake-up on Voltage Change

The “wake up on voltage change” is perhaps one of the most useful trigger wake-up triggers. The electrical system of the vehicle produces a steady voltage, when the vehicle is at rest. However, there are many things that can produce a measurable dip — the driver pushes the “unlock” button on the car remote, opens a door, or cranks the engine — that can be used to wake up the device.

We will use a resistor to simulate a dip, to bring the device out of the sleep state.

Here is the complete procedure for this portion of the lab:

1. Issue the STVR command to read the current voltage
2. Connect the 100k resistor between the ANALOG_IN and GND pins of the STN1170 module



3. Read the new voltage (STVR). It should be roughly 1 volt less than what you measured in step #1
4. Enter the following commands:

`STSLVG on`	*wake on voltage change*
`ATZ`	*reboot to activate*
`STSLCS`	*confirm the trigger is active*
`STSLEEP`	*put device to sleep*

5. Briefly connect the 100k resistor between the ANALOG_IN and GND pins, and observe the STN1170 wake up and print the welcome prompt.

LAB 3:
OBD Development and Testing

Purpose:

To gain experience configuring the OBD simulator to aid development and testing.

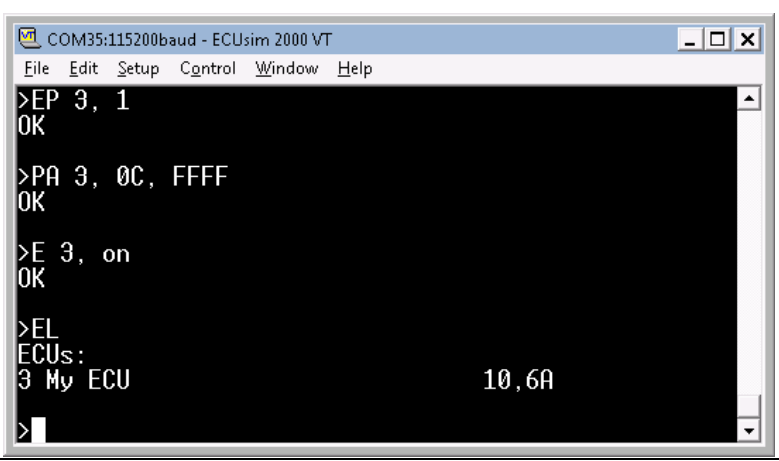
Overview
In this lab, you will create and configure a virtual ECU, add a PID, and create a custom fault set.

Creating a basic ECU

Enter the following commands in the “OBD Simulator” TeraTerm window:

```
SP 1                set the protocol to PWM
EDA                 delete all existing (default) ECUs
EA 3                create ECU #3
EN 3, “My ECU”      specify ECU name
EAP 3, 10           assign ECU physical address $10
EAF 3, 6A           assign ECU functional address $6A
EP 3, 1             set ECU’s protocol preset to PWM
PA 3, 0C, 0FA0      add PID (RPM = 1000)
E 3, on             turn on the ECU
```

Send the EL (“list ECUs”) command to verify that the ECU has been created:

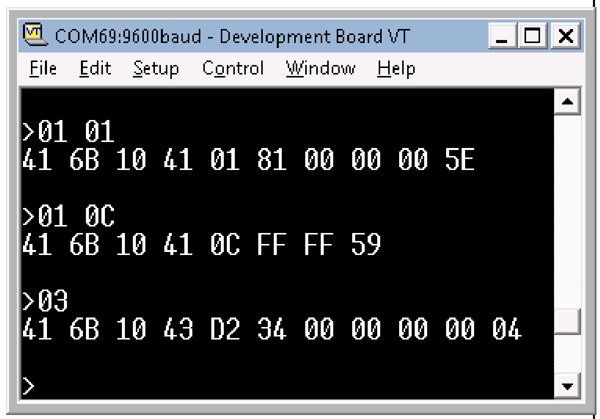


Now, let's add a simple fault set:

```
DSA 3, 1, U1234    report a stored DTC (U1234) after a fault event
PAUDC 3, on        enable automatic updates of stored DTC count
PAUMS 3, on        enable automatic updates of MIL status
```

All that remains, is to test the newly created setup, by entering the following commands in the "Development Board" TeraTerm window:

```
01 01        MIL status & DTC count
01 0C        RPM value
03           stored DTCs
```



From the responses, you can see that the MIL is set, there is one stored DTC (U1234), and RPM is at its maximum value.

Besides obvious convenience, an OBD simulator gives the developer the ability to set PIDs to arbitrary values outside the "usual" value range, or even have the virtual ECU send an invalid response (e.g., RPM reported as a 3-byte value). Both are useful for stress-testing OBD software in the lab.

Reference Material

- **SAE Standards** (sae.org): J1979, J1850, J2012, J1939
- **ISO Standards** (iso.org): ISO 9141, ISO 14230, ISO 15765
- **OBD Software Development Tutorials**
  http://www.obdsol.com/articles/
- **ECUsim 2000 User Guide**
  http://www.scantool.net/scantool/downloads/101/ecusim_2000-ug.pdf
- **ECUsim 2000/5100 Programming Manual**
  http://www.scantool.net/static/documentation/ecusim/ecusim-pm.pdf
- **STN1100 Family Reference and Programming Manual**
  http://www.scantool.net/scantool/downloads/98/stn1100-frpm.pdf