

7.5.

Postavljanje scapy konzole za korištenje XCP-a:

```
load_layer('can')
load_contrib('cansocket')
load_contrib('automotive.xcp')
load_contrib('automotive.xcp.xcp')
sock = CANSocket(bustype='socketcan', channel='vcan0', basecls=XCP0nCAN)
```

Slanje connect paketa

```
>>> pkt = XCP0nCAN(identifier=0x700) / CT0Request() / Connect()
>>> pkt
<XCP0nCAN identifier=0x700 |<CT0Request pid=CONNECT |<Connect |>>>
>>> sock.send(pkt)
16
```

8.5.

<https://piembsystech.com/xcp-protocol/>

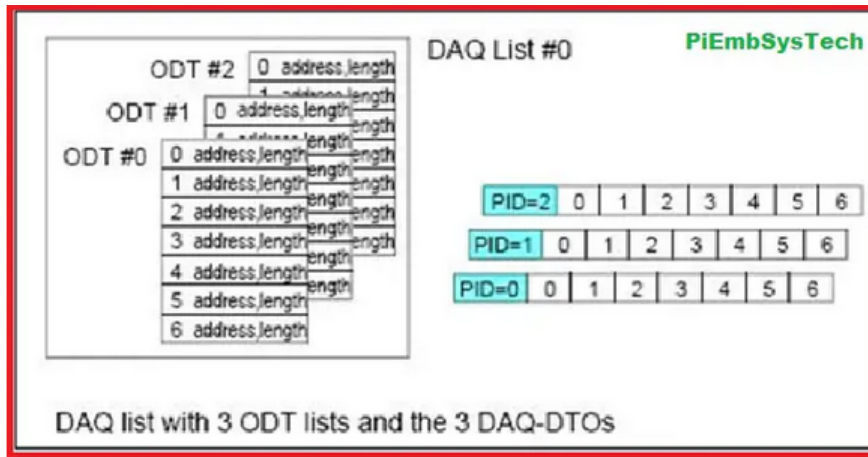
- CTO - poruke
 - CMD/RES/ERR/EV/SERV
- DTO - podaci
 - DAQ/STIM

The DTO has two types of objects that are both used for event-driven reading of variables from, or writing values to, the memory of the slave.

1. DAQ (Data Acquisition).
2. STIM (Stimulation).

Data Acquisition (DAQ): A core feature of the XCP protocol is the DAQ lists. In order to be able to send a large amount of data in a small amount of time and with low bandwidth load desirable, XCP offers the ability to configure lists that take care of the transmitting requested data at a given interval. Each DAQ list (Figure 10) has a number of Object Descriptor Tables (ODTs) that in turn contain Object Descriptor Table Entries (ODT Entries) as described in Figure 11. Each the ODT Entry has an address and a length, these make out the description of the parameter that it represents. When the DAQ list has processed the contents of the list are copied to the corresponding address of each entry in each ODT. The slave doesn't receive an acknowledgment that the master

has received the data correctly.



XCP DAQ Diagram

preostalo

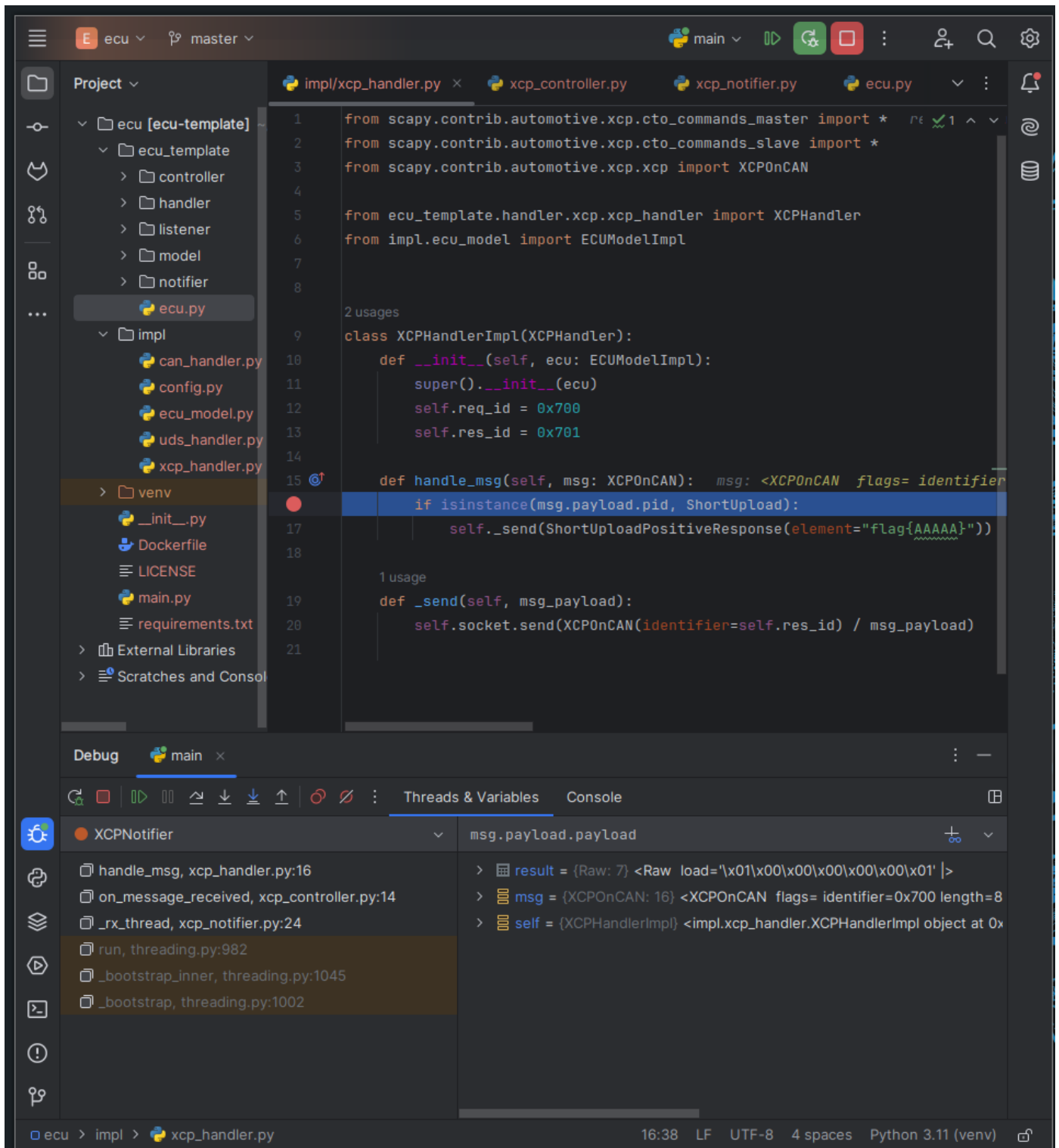
- driver
 - dodati opciju za povezivanje dockera u namespace na defaultni
 - refaktoriraj driver.go u manje funkcije
 - perzistentnost
 - upakirat za linux distribucije
 - github actions za deploy na dockerhub
 - brisanje host interfeasa nakon brisanja mreze
 - podizanje host interfeasa nakon stvaranja mreze
 - upload na dockerhub
- zadaci
 - napraviti repo koji builda zadatke za dockerhub automatski s github actionsima
 - XCP - dump memory caring caribou
 - UDS Authentication
 - CAN zadatak s dva ECU-a i GUI-jem
 - uds routine control bi se mogao koristiti za ovo
 - upalit zmigavce
 - postic vecu brzinu nego maksimalnu
 - kad uspije, postavit flag na sabirnicu
 - UDS security access MITM izmedju dvije sabirnice
 - UDS zadatak s GUI-jem, routine control
- ecu_template
 - koristit import lib pa po pristunosti datoteka dodati ili maknut neke protokole
 - dodati template za UDS SA
 - povezat s nekim GUI-jem
 - KUKSA server
 - dodati glavni program za slanje repetitivnih poruka umjesto CAN_BCM-a?
 - README
 - čemu svaka od impl datoteka služi
 - kompatibilnost s caring caribouom
 - primjeri kako neke stvari implementirati
 - tagovi za razne varijante templateova
 - XCP
 - DoIP

- parsiranje CAN signala po DBC-u
- dodatni programi:
 - gw program za povezivanje vise dockercan mreza?
 - program za cannelloni udaljeni pristup
- skripta za generiranje docker composeova

10.5.

Iz nekog razloga scapy ne prepoznaje automatski shortupload paket po primitku.

```
)>> pkt2 = XCP0nCAN(identifier=0x700) / CT0Request() / ShortUpload(nr_of_data_elements=1, address=0x0)
>>> pkt2
<XCP0nCAN identifier=0x700 |<CT0Request pid=SHORT_UPLOAD |<ShortUpload nr_of_data_elements=1 address=1
|>>> sock.srl(pkt2)
Begin emission:
Finished sending 1 packets.
^C
Received 0 packets, got 0 answers, remaining 1 packets
>>> sock.srl(pkt2)
Begin emission:
Finished sending 1 packets.
```



- scapy poslani CTORrequest prepoznaje kao CTORresponse?

```

File "console", line 1
>>> load_layer('can')
^^
SyntaxError: invalid syntax
>>> load_layer('can')
File "console", line 1
>>> load_layer('can')
^^
SyntaxError: invalid syntax
>>> load_contrib('cansocket')
File "console", line 1
>>> load_contrib('cansocket')
^^
SyntaxError: invalid syntax
>>> load_contrib('automotive.xcp')
File "console", line 1
>>> load_contrib('automotive.xcp')
^^
SyntaxError: invalid syntax
>>> sock = CANSocket(bustype='socketcan', channel='vcan0', basecls=XCPOnCAN)
KeyboardInterrupt
>>>
KeyboardInterrupt
>>> load_layer('can')
>>> load_layer('can')
>>> load_contrib('cansocket')
INFO: Configuration 'conf.contribs['CANSocket'] not found.
INFO: Using native CANSocket.
Specify 'conf.contribs['CANSocket'] = {'use-python-can': True}' to enable python-can CANSocket.
>>> load_contrib('automotive.xcp')
>>> sock = CANSocket(bustype='socketcan', channel='vcan0', basecls=XCPOnCAN)
Traceback (most recent call last):
  File "console", line 1, in <module>
NameError: name 'XCPOnCAN' is not defined
>>> load_contrib('automotive.xcp')
>>> load_contrib('automotive.xcp')
KeyboardInterrupt
>>> sock = CANSocket(bustype='socketcan', channel='vcan0', basecls=XCPOnCAN)
>>> sock
<scapy.contrib.cansocket_native.NativeCANSocket object at 0x739b7d42f1d0>
>>> pkt = XCPOnCAN(identifier=0x700) / CTORRequest() / Connect()
>>> sock.send(pkt)
16
>>> sock.send(pkt)
16
d='\x00'.send(XCPOnCAN(identifier=0x700)/CTORResponse())/Raw(load=...)
...
KeyboardInterrupt
>>> sock.send(XCPOnCAN(identifier=0x700)/CTORResponse())/Raw(load='\x00')
16
>>>

```

```

) python3 -m scapy
INFO: Can't import PyX. Won't be able to use psdump() or pdfdump().
WARNING: Python not available. Using standard Python shell instead.
AutoCompletion, History are disabled.
SYNOPSIS
P /SCS/CSX  ACS  Welcome to Scapy
              AC  Version 2.5.0
A/PS        /SPPS
YP          /SC  https://github.com/secdev/scapy
SPS/A.     SC
Y/PACC     PP  Have fun!
PY+AYC     CAA
          YCYC/SCYP
>>> load_layer('can')
>>> load_contrib('cansocket')
INFO: Configuration 'conf.contribs['CANSocket'] not found.
INFO: Using native CANSocket.
Specify 'conf.contribs['CANSocket'] = {'use-python-can': True}' to enable python-can CANSocket.
>>> load_contrib('automotive.xcp')
>>> load_contrib('automotive.xcp.xcp')
>>> sock = CANSocket(bustype='socketcan', channel='vcan0', basecls=XCPOnCAN)
>>> rec = sock.recv()
>>> rec
<XCPOnCAN flags= identifier=0x700 length=2 reserved=0 |<CTORResponse packet_code=RES |<Raw load='\x00' |>>
>>> rec
<XCPOnCAN flags= identifier=0x700 length=2 reserved=0 |<CTORResponse packet_code=RES |<Raw load='\x00' |>>
>>> rec = sock.recv()
>>> rec
<XCPOnCAN flags= identifier=0x700 length=2 reserved=0 |<CTORResponse packet_code=RES |<Raw load='\x00' |>>
>>>

```

```

> candump vcan0
vcan0 700 [2] FF 00
vcan0 700 [2] FF 00

```

Sadržaj paketa za CTOResponse i CTORrequest se ne razlikuje već je kontekstualan. Scapy automatski sve što pročita smatra Responseom, obzirom da je namijenjen za komunikaciju s ECU-ovima.

```

> candump vcan0
vcan0 700 [2] FF 00
vcan0 700 [2] FF 00

```