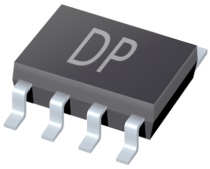
FPC



DevicePlugin

Vector CAN(FD)/LIN

Programmer's Manual

Revision 2022.06.16

Table of Contents

1 Description

Implementation of Vector's XL Driver Library. It enables to control CAN and LIN functionality using corresponding Vector's piggies on CANboardXL PCI card.

2 Commands

2.1 *idn? (Identification)

```
*idn?
```

Gets the plug-in identification string.

Parameters

No parameters.

Return value

The identification string in standard format "<company>,<product/name>,<serial-no>,<version>".

2.2 accept (Set acceptance filter)

```
accept: <definition>
```

Set ID-based acceptance filter of receiving messages.

Parameters

definition	[string]	A comma-separated list of ID ranges in specified format. **Format:** {!}<id-from>{-<id-to>} **Where:** • "! " - if present, the range is excluded, otherwise included • id-from - first value of range • id-to - last value of range **Note:** IDs can be in hexadecimal ('0x' prefix), binary ('0b' prefix) or in decimal format Using "all" keyword is the filter cleared (= accept all).

Return value

No return value

Example

accept: 50,0x100
Accept only messages with IDs 50 and 0x100 (= 256 decimal).

accept: 30-40
Accept message with IDs in range 30 to 40.

accept: ! 30-40

Reject messages with IDs in range 30 to 40. All others are accepted.

`accept:30-40,100-120`
Accept two ranges of IDs: 30 to 40 and 100 to 120. All others are rejected.

`accept:!100-120,!0xff`
Reject messages in range of 100 to 120 and 0xFF. All others are accepted.

`accept:all`
Clear acceptance filter = accept all.

2.3 rx (Receive message)

```
rx:read?{:<timeout>{:<format>}}
rx:readls?{:<timeout>{:<format>}}
```

Receive message from the receiving buffer. The `read?` reads the first received message (the oldest) and `readls?` reads last one received (the newest). Received message **stays in the buffer until it is read**. Before using this command, the accept command to set an acceptance filter should be used. Otherwise (depending on activity on the bus) in the buffer will be all received message and it will probably take too many cycles to read expected message.

Parameters

`timeout`	[int]	Timeout in [ms] to wait for a message to be received. **Default:** 1000ms
`format`	[enum]	Return format of message's data bytes. Possible options: • `hex` (always 2-digit, `00` to `FF`, no prefix) • `bin` (always 8-digit, `00000000` to `11111111`, no prefix) • `dec` (always 3-digit, `000` to `255`, no prefix) **Default:** `hex`

Return value

A) if at least one message is received until timeout will occur:
Message ID and data-bytes sequence in format:
`<id>:<db0><db1>...<db`$_N$`>`
Where:
`id` - fixed length ID of received message (LIN = 2 digit, CAN = 3 digit), always in hexadecimal
`db`$_N$ - message's data-bytes with no separators or any space between each other, fixed lengths

ID is separated by colon from data-bytes.

B) if no message is received in specified timeout, the function returns:
`TIMEOUT`

Example

`rx:read?`
Returns first message with defalt 1000ms timeout and hexadecimal format:
i.e. `100:20406080aabbccdd`
=> ID = 0x100, data bytes = 0x20, 0x40, 0x60, 0x80, 0xAA, 0xBB, 0xCC, 0xDD
After this command is proceeded, this returned message is removed from the receive queue and cannot be read again, until new-one is received from the bus.

`rx: readls?: 2000: bin`
Returns last message with user-specified 2000ms timeout and binary return format:
i.e. 100:00100000010000001010101010111011
=> ID = 0x100, data bytes = 0b00100000, 0b01000000, 0b10101010, 0b10111011

```
rx: count?
```

Return number of unread messages.

Parameters

No parameters.

Return value

Number of messages (integer, decimal format).

```
rx: clr
```

Clear all received messages from receiving buffer.

Parameters

No parameters.

Return value

No return value.

2.4 close (Close a channel)

```
close
```

Close the currently opened LIN or CAN channel.

Parameters

No parameters.

Return value

No return value.

2.5 LIN-specified

2.5.1 open (Open the LIN channel)

```
open: { mode=[ enum] }{ ; bitrate=[ int] }{ ; ver=[ enum] }{ ; dlc=[ string-array] }
      { ; sla-tasks=[ int-array] }
```

Open the LIN channel using specified bitrate and LIN version and set DLC table and activates specified master's slave tasks.

Parameters

`mode`	[enum]	LIN mode: • `master` • `slave`

		Default: master
`bitrate`	[int]	LIN bitrate in [bps], typical values: • `2400` • `9600` • `14400` • `19200` **Default:** `19200`
`ver`	[enum]	LIN version, possible values: • `V13` (1.3) • `V20` (2.0) • `V21` (2.1) **Default:** `V20`
`dlc`	[string-array]	Specifies expected length of messages to be received. This is required because the LIN frame does not contain a data-length field. Format: "$<id_0>/<dlc_0>,<id_1>/dlc_1>,...,<id_N>/<dlc_N>$" Where id_N is the LIN ID and dlc_N is expected message length of this ID to be received. ID can be in hexadecimal (with '0x' prefix) or decimal format. Length must be in decimal format in range 1 to 8. **Default:** 8 bytes for all non-specified IDs here
`sla-tasks`	[int-array]	LIN master can only send a master's ID request, without any data. To send data from the master, the only way is to activate "master's slave task" functionality: this is the regular LIN slave, but controlled by master. When master send an ID request and this ID has a "slave task" activated, this task will immediately response by its data => resulting data are sent by the master. This argument is required while in SLAVE mode, at least one response frame should be defined. The slave automatically responses on bus by associated message. Using this argument it is possible to set IDs, for which the master's slave task will be activated (using only these IDs the master can send data).

Return value

No return value

Example

`open: dlc="10/4,0x20/6,50/2";sla-tasks="12,14"`
Opens the LIN channel in MASTER mode, set expected message's DLCs (ID 10 = 4 bytes, ID 32 = 6 bytes, ID 50 = 2 bytes) and activate master's slave tasks for IDs 12 and 14.

`open: mode=slave;sla-tasks="10,12"`
Opens the LIN channel in SLAVE mode and set ID 10 and 12 for response of this slave node.

2.5.2 sla-msg (Manage predefied LIN messages)

`sla-msg: clr`

Clear all predefined LIN messages.

Parameters

No parameters

Return value

No return value

`sla-msg: set: <alias>: <db0>{: <db1>: ... <dbN>}`

Add new or modify existing predefined LIN message. These predefined message can be used by req LIN command.

Parameters

`alias`	[string]	LIN message alias. You can use digits, letters, dash and underscore. Alias cannot start by a digit (0-9). If specified alias does not exist, the new message with this alias is created. Otherwise the data of existing message will be modified.
db_N	[int]	Message data bytes 0 to 7. At least 1 data byte must be passed. Maximum is 8 bytes. Range: 0x00 to 0xFF (0 to 255)

Return value

No return value

Example

`sla-msg: clear`
Clear all existing messages from internal table.

`sla-msg: set: my-packet: 0x10: 0x20: 0x30: 0x40`
Add a new message with alias 'my-packet' and 4-byte data sequence 0x10, 0x20, 0x30 and 0x40.

`sla-msg: set: my-packet: 0x60: 0x70`
Modify data of existing message with 'my-packet' alias to only 2-byte sequence 0x60 and 0x70.

2.5.3 sla-task (Set slave task data)

`sla-task: <id>: <db0>{: <db1>: ... <dbN>}`

Modify the slave task's data of specified LIN ID.

Parameters

`id`	[int]	LIN identifier (6-bit). **Range:** 0x00 to 0x3F (0 to 63)
db_N	[int]	Message data bytes 0 to 7. At least 1 data byte must be passed if message for specified ID is required. Maximum is 8 bytes.

Return value

No return value

Example

`sla-task: 10: 0x10: 0x20: 0x30: 0x40`

Modifies the slave task's data with ID=10 to 0x10, 0x20, 0x30 and 0x40.

2.5.4 req (Send master's LIN request)

```
req: <id>{: <msg-alias>}
req: <id>{: <db0>{: <db1>: ... <dbN>}}
```

Sends the master's LIN request with specified LIN ID. If the master's slave task is activated for specified ID, the data are required - specified by msg-alias or raw data.

Parameters

`id`	[int]	LIN identifier (6-bit). **Range:** 0x00 to 0x3F (0 to 63)
db_N	[int]	Message data bytes 0 to 7. At least 1 data byte must be passed if message for specified ID is required. Maximum is 8 bytes.

Return value

No return value

Example

`req: 10`
Send master's LIN request using ID = 10 (decimal).

`req: 12: 0x11: 0x22: 0x33: 0x44: 0x55: 0x66`
Send master's LIN request using ID = 12. Because for this ID was activated the slave task using open command (see examples), the data are required - in this example sequence of 6 bytes.

`req: 14: my-packet`
Near the same like previous example, but the data are specified by alias of predefined LIN message (see examples of sla-msg command).

2.6 CAN-specified

CAN (Controller Area Network)
CAN-FD (Controller Area Network - Flexible data rate)

2.6.1 open (Open the CAN channel)

```
open: {bitrate=[int]}{: dataBitrate=[int]}{: samplePoint=[int]}{: crcMode=[enum]}
```

Open the CAN channel using specified bitrate.

Parameters

`bitrate`	[int]	CAN bitrate in [bps], range 15000 to 1000000, typical values: • `20000` (20 kbit/s) • `33300` • `50000` • `62500` • `83300` • `100000` (100 kbit/s) • `125000` • `250000` • `500000` (0,5 Mbit/s)

		• `1000000` (1,0 Mbit/s) **Default:** `500000`
`dataBitrate`	[int]	Data (CANFD) bitrate in [bps] which must be equal or greather than `bitrate` parameter, typical values: • `20000` (20 kbit/s) • `33300` • `50000` • `62500` • `83300` • `100000` (100 kbit/s) • `125000` • `250000` • `500000` (0,5 Mbit/s) • `1000000` (1,0 Mbit/s) • `2000000` (2,0 Mbit/s) • `5000000` (5,0 Mbit/s) • `8000000` (8,0 Mbit/s) **Default:** `500000` * Only for CAN-FD capable device.
`samplePoint`	[int]	Bit sample point in percent of bit width. • `40` • `60` • `65` • `70` • `75` **Default:** 65% * Only for CAN-FD capable device.
`crcMode`	[enum]	**Default:** ISO * Only for CAN-FD capable device.

Return value

No return value

Example

`open: bitrate=1000000`
Opens the CAN channel @ 1,0 Mbit/s.

`open: bitrate=500000: dataBitrate=1000000: crcMode=iso: samplePoint=65`
Opens the CAN-FD channel with arbiter bitrate 500,0 kbit/s, payload bitrate 1,0 Mbit/s, default crc mode given by ISO with bit sample point at 65%.

2.6.2 set (Set default command parameters)

```
set{:rxTimeout=<int>}{:txTimeout=<int>}{:idfmt=<enum>}{:fd=<bool>}
```

Set the default parameters for tx/rx commands.

Parameters

`txTimeout`	[int]	Default timeout value for tx command. **Default:** 1000ms **Range:** 0 to 30000ms
`rxTimeout`	[int]	Default timeout value for rx command. **Default:** 1000ms

		Range: 0 to 30000ms
`idfmt`	[enum]	Default value for message ID format for tx command. **Values:** • Auto - automatic resolution of standard format or extended for given message ID • Std - force using of 11-bit message ID format • Ext - force using of 29-bit message ID format **Default:** Auto
`fd`	[bool]	Enable or disable transmitting of CAN-FD frame, when the device is capable of. If disabled, maximum payload size is 8byte and data bitrate is not used. If enabled, maximum payload size is 64byte and data bitrate can be used. **Default:** false * Only for CAN-FD capable device.

Return value

No return value.

Example

`set:rxTimeout=1000:txTimeout=1000:idfmt=auto:fd=false`
Set default timeout value for CAN read operation to 1second. For CAN TX operation to 1second, default ID format type to automatic and turns off flexible data rate if card is capable of.

2.6.3 tx (Transmit a CAN message)

`tx:<id>{:<db0>{:<db1>:...<dbN>}}{:timeout=<int>}{:idfmt=<enum>}{:fd=<bool>}`
`tx:<id>{:<db0>{:<db1>:...<dbN>}}{:tx:<id>{:<db0>{:<db1>:...<dbN>}}}{:...}{:timeout=<int>}{:idfmt=<enum>}{:fd=<bool>}`

Transmit a CAN message(s).
Some devices (e.g. VN1530) can be capable of handling CANFD message(s).

Parameters

`id`	[int]	CAN message identifier. Correct range is automatically picked, when it is not forced by argument `ext`. **Standard 11bit range:** 0x000 to 0x7FF (0 to 2047) **Extended 29bit range:** 0x00000000 to 0x1FFFFFFF
db_N	[int]	Message data bytes. **CAN:** maximum is 8 bytes. **CANFD:** maximum is 64 bytes.
`timeout`	[int]	Waits for acknowledge the message(s) by any other device on the bus. When no acknowledgment was done, the Vector card will use almost 100% bus time to try transmit this message and no other messages can be transmitted until the message is delivered or canceled by calling commands reset or close. If timeout argument is 0, the function does not wait for result! **Range:** 0 to 30000ms **Default:** using the default `txTimeout` value from command set. *This parameter is relevant for all messages in this command.*
`idfmt`	[enum]	Default value for message ID format for tx command.

		Default: using the default `idFmt` value from command set. *This parameter is relevant for all messages in this command*
`fd`	[bool]	Enable or disable transmitting of CAN-FD frame, when the device is capable of. **Default:** using the default `fd` value from command set. *This parameter is relevant for all messages in this command*

Return value

No return value.

Example

`tx:100:0x1a:0x2b:0x3c:0x4d`
Send a CAN message with ID = 100 (decimal) and 4 data-byte sequence (hex) 0x1A, 0x2B, 0x3C, 0x4D

`tx:0x100:0x1a:0x2b:0x3c:0x4d:tx:0x200:0xFF`
Send a two CAN messages: ID = 0x100 (decimal) and 4 data-byte sequence (hex) 0x1A, 0x2B, 0x3C, 0x4D and after that ID = 0x200 (decimal) and 1 data-byte sequence (hex) 0xFF

2.6.4 reset (Reset opened bus)

```
reset
```

Performs reset of the CAN channel and rx/tx error counters, then it clear the receive and transmit queues. While reset the channel is turned off and on again and therefore any message on the bus will be lost! Reset does not reset whole configuration, therefore this is faster way than commands close and open. Usable only when is channel opened.

Parameters

No parameters

Return value

No return value

2.6.5 state? (Query bus state)

```
state?
```

Query the internal Vector bus state.

Parameters

No parameters

Return value

- `OFF` - bus was not opened
- `PASSIVE` - internal tx or rx error counter reach 128, in this state device may not receive any data from CAN bus
- `WARNING` - internal tx or rx error counter is between 1 and 127
- `ACTIVE` - normal bus operation

State PASSIVE or OFF may spontaneously happen:

- After too much retransmit tries (tx frame not acknowledged by anyone more than 128x).
- HW problem on bus (missing termination resistors, bus overload).

- Card internal buffer overrun (computer was not fast enough)