Using the emtas binary CANopen examples

Version: v.1.0

First releas

		Date	signature
creation	Heinz-Jürgen Oertel	10/07/13	
check			
release			

Versionen

Version	changes	Date	Bearbeiter	release
1.0	First release	11/07/2013	oe	ged

Content

1 Introduction

To evaluate CANopen emtas provides ready to use compiled examples for different platforms. Each chapter briefly describes the example and how to communicate with it using a simple CAN analyzer. It is supposed that you own already a CAN interface and at least a simple CAN analyzer which is able to display what happens on the CAN bus and which is able to send out user specified CAN frames. To test the provided CANopen binaries we use simple CAN commands for testing SDO, node monitoring via the CANopen Heartbeat mechanism, PDO and EMCY communication.

It is recommended that you own the CiA 301 specification in order to get an exact explanation what is going on with all the bits and bytes within the CANopen protocol CAN frames. If data types greater than one byte are used, remember that the CANopen byte order on the CAN bus is low byte first (little endian).

In order to be tested with standard CANopen configuration tools like emtas CDE, the device EDS file (electronic data sheet) is always part of the provided zip or tgz archive. Depending on the target platform the example is delivered as executable, HEX or ELF file.

The default CANopen node-id used in the examples is 32, or hex 0x20. This is important to know because a lot of the default communication frames are derived from the CANopen node-id, as is the boot-up message, which is 0x700+node-id and results in 0x720.

If only a basic analyzer is available for testing advice is given how to build CANopen requests by assembling the CAN id with appropriate data bytes.

To become more familiar with CANopen and the provided examples it is recommended to use a CANopen configuration tool which is able to read the device EDS File. Emtas offers an evaluation version of its CANopen Device Explorer, which requires an installed CAN hardware interface.

1.1 Used CAN format description

Can frames, whether received or composed, are given on a single line starting with the CAN frame id followed by the data bytes.

This is the boot-up frame:

0x0720	0x7f							

Green as background is used for frames sent by the CANopen device, red for tables showing what has to be entered in a transmit formular of the analyzer or equivalent CAN tool.

1.2 Basic example behavior

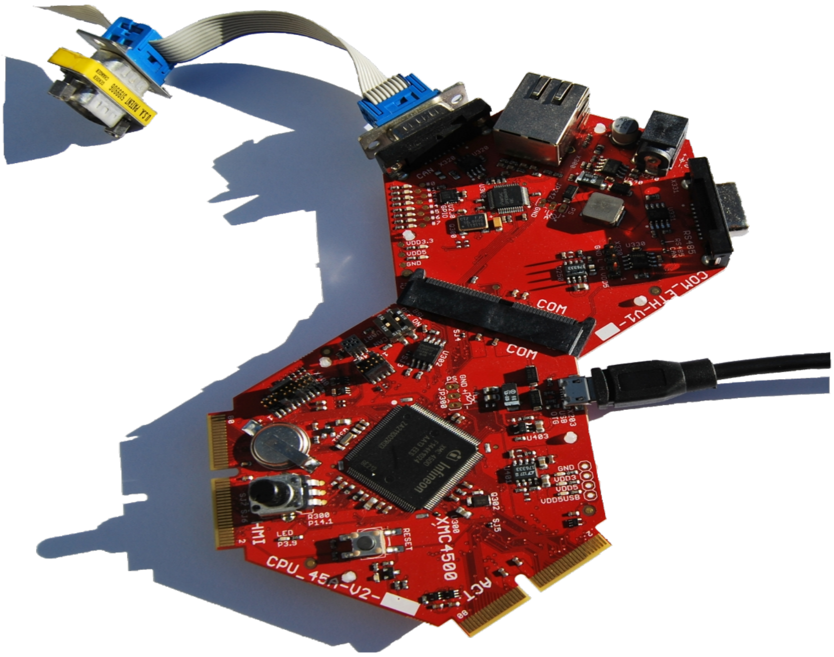
The examples implement the CiA 401 profile, a profile describing generic process I/O modules. It depends what I/O ports are available on the chosen target boards that is available for testing.

The default bit rate used is 250Kbit/s, the CANopen Node-ID is 32.

The examples do not send their Heartbeat messages automatically. It has to be configured via SDO requests to do so, in order to have not so much frames on the analyzer display.

2 Infineon Hexagon Kit

The Infineon Hexagon kit comes with the XMC4500 ARM cortex-M4 based controller which implements the MultiCAN module. If the Kit is complemented with the communication module COM_ETH the Kit is extended with CAN transceiver and a DSUB9 based CAN connector. Additional the board has eight LEDs which are controlled by the example using the CiA 401 device profile.



2.1 *Download and install the demo application*

The zip archive contains the slave401.elf and slave401.hex file. Whether the Dave IDE can be used to download the ELF file or any Flash utility, like the Segger J-Flash can be used to download the hex file to the Hexagon board.

http://www.segger.com/admin/uploads/productDocs/UM08003_JFlashARM.pdf

2.2 *Running the tests*

The used analyzer should be connected to the device before starting the application. If the device starts up, or the boards reset button is pushed, the CANopen boot-up message is sent. It is a frame with the CAN frame id 0x700 + node-id.

0x0720	0x00							

Our first test will be reading out object 0x1000.

0x0620	0x40	0x00	0x10	0x00	0x00	0x00	0x00	0x00

And we get the answer:

0x05a0	0x43	0x00	0x10	0x00	0x91	0x01	0x00	0x00

The last 4 byte contain the SDO response value. Because this object is of type UNSIGNED32 the read value is 0x00000191, decimal 401, the CANopen profile number.

As next step we will read out the Heartbeat producer time.

0x0620	0x40	0x17	0x10	0x00	0x00	0x00	0x00	0x00

0x05a0	0x4b	0x17	0x10	0x00	0x00	0x00	0x00	0x00

The last 4 byte contain the SDO response value. Because this object is of type UNSIGNED16 read value is zero. Now try to change the Heartbeat Producer Time to 1s. In CANopen the value has to be given in ms. We send the SDO request with an hex value of 0x3e8:

0x0620	0x2b	0x17	0x10	0x00	0xe8	0x03	0x00	0x00

with the low byte in byte 5 and the high byte in byte 6. The device should response with:

0x05a0	0x60	0x17	0x10	0x00	0x00	0x00	0x00	0x00

And on the Analyser screen a repeated Heartbeat message should be displayed:

0x0720	0x7F							

The one and only data byte 0x7f shows the CANopen communication state of the device. 0x7f is PRE-OPERATIONAL.

We can now switch the device to the state OPERATIONAL by means of a single NMT message:

0x0000	0x01	0x20						

The fist byte of the NMT message is the NMT command, switch to OPERATIONAL, the second the node address, 0x20 decimal 32, our node's node-id.

The one byte content of the Heartbeat message with CAN id 0x0720 changes now to 0x05 which is the code for NMT state OPERATIONAL.

0x0720	0x05							

While switching into state OPERATIONAL the device will send out two configured TPDOs.

0x01a0	0x00	0x00	0x00					

0x04a0	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00

The content of all bytes is zero. The example does not have digital and analog inputs available on the hexagon.

Now you should again disable the Heartbeat Producer by writing zero to the Heartbeat object:

0x0620	0x2b	0x17	0x10	0x00	0x00	0x00	0x00	0x00

The device should response with:

0x05a0	0x60	0x17	0x10	0x00	0x00	0x00	0x00	0x00

and now Heartbeat messages disappear from the analyzer display.

The device is still OPERATIONAL. Because the example is using the 8 LEDs on the COM expansion board as digital outputs we can control these LEDs now by use of a PDO. The CAN id of the first RPDO of the device is 0x0220 (0x200 + 32). This RPDO has three mapped objects, each one byte. Therefore we have to send 3 bytes with this PDO but only the first one is really mapped to the output of the LEDs.

0x0220	0xaa	0x00	0x00					

This will switch on every second LED. Sending:

0x0220	0x55	0x00	0x00					

will switch on the odd numbered LEDs instead.

Last we can show how the CANopen device reacts on a wrongly configured RPDO. As already explained the RPDO1 is configured to receive three bytes. When we send the PDO with only two bytes.

0x0220	0xaa	0x55						

we get an EMCY message sent out by the device

0x00a0	0x10	0x82	0x	0x00	0x00	0x00	0x00	0x00

The CAN id 0xa0 is composed of 0x80 + node-id, the first and second byte contain the two byte error code, always remember the little endian byte order. The value of the error code is 0x8210, which according CiA 301 is "PDO not processed due to length error". The third byte is the content of the device Error Register in Object 0x1001.

To get more familiar with CANopen it is recommended to use a CANopen configuration tool which is able to read the device EDS File. Try to read out object 0y1008, "Manufacturer device name".

3 Linux (Desktop x86)

PCs having a CAN Interface, whether a PCI or USB interface, supported by either the can4linux or SocketCAN device driver can be used.

3.1 *Download and install the demo application*

There is nothing special on the PC. Download the tgz archive linux_x86.tgz and select the executable suited for your target. The archive contains executables compiled with can4linux or SocketCAN support and for 32 or 64 bit systems. Before testing the CANopen example, please make sure that the used SocketCAN driver is working by testing the CAN utilities like `cansend` and `candump` (with interface can0). Or in case of can4linux use the `can_send` and `receive` applications.

The selected executable can be stored everywhere in the file system. It is a command line application. Its behavior can be controlled by command line switches. Once started information on the node state are displayed on *stdout.*

The can4linux application `slave1_*_can4linux` is using `/dev/can0` and 250Kbit/s as default. This can be changed by command line switches -b and -D.

The SocketCAN application `slave1_*_socketcan` is using the network device `can0` by default. SocketCAN requires, that the interface is configured by the sysadmin before any application can use it. Use following commands to set the interface up and running:

```
ip link set can0 type can bitrate 250000 sjw 1
ip link set can0 up
```

3.2 *Running the tests*

On the Linux systems all process IO if any are only simulated. The CiA 401 device consists of 3 digital out ports at object 0x6200:1 to 0x6200:3. With objects 0x6202:1 to 0x6202:3 the simulated output polarity can be changed (see CiA 401). If one of the output port values is changed via CANopen SDO or PDO, the example will print a status line at stdout.. The example also simulates the CANopen indicator LEDS. It will display a green and, in case of errors, a red character 'O'. The green simulated LED will display the nodes NMT slave status. Each status change is also reported by displaying a status line with this information.

The used analyzer should be connected to the device before starting the application. If the device starts up, the CANopen boot-up message is sent. It is a frame with the CAN frame id 0x700 + node-id followed by an EMCY (0x80 + node-id).

0x0720	0x00							

0x00a0	0x34	0x12	0x01	0x01	0x02	0x03	0x04	0x05

Our first test will be reading out object 0x1000.

0x0620	0x40	0x00	0x10	0x00	0x00	0x00	0x00	0x00

And we get the answer:

0x05a0	0x43	0x00	0x10	0x00	0x91	0x01	0x82	0x00

The last 4 byte contain the SDO response value. Because this object is of type UNSIGNED32 the read value is 0x00820191. The last significant two bytes represent the CANopen profile number implemented. It is 0x0191, decimal 401, the CANopen profile number. For the most significant bytes, see CiA 401.

As next step we will read out the Heartbeat producer time.

0x0620	0x40	0x17	0x10	0x00	0x00	0x00	0x00	0x00

0x05a0	0x4b	0x17	0x10	0x00	0x00	0x00	0x00	0x00

The last 4 byte contain the SDO response value. Because this object is of type UNSIGNED16 read value is zero. Now try to change the Heartbeat Producer Time to 1s. In CANopen the value has to be given in ms. We send the SDO request with an hex value of 0x3e8:

0x0620	0x2b	0x17	0x10	0x00	0xe8	0x03	0x00	0x00

with the low byte in byte 5 and the high byte in byte 6. The device should response with:

0x05a0	0x60	0x17	0x10	0x00	0x00	0x00	0x00	0x00

And on the Analyser screen a repeated Heartbeat message should be displayed:

0x0720	0x7F							

The one and only data byte 0x7f shows the CANopen communication state of the device. 0x7f is PRE-OPERATIONAL.

We can now switch the device to the state OPERATIONAL by means of a single NMT message:

0x0000	0x01	0x20						

The fist byte of the NMT message is the NMT command, switch to OPERATIONAL, the second the node address, 0x20 decimal 32, our node's node-id.

The one byte content of the Heartbeat message with CAN id 0x0720 changes now to 0x05 which is the code for NMT state OPERATIONAL. While switching into state OPERATIONAL the device will send the configured TPDO.

0x0720	0x05							

0x01a0	0xe7	0x0d	0xa1	0x05				

TPDO1 of the device has mapped two counters, each UNSIGNED16, therefore the PDO has 4 byte. The value of the counter is of course a random one at time switching to OPERATIONAL.

Now you should again disable the Heartbeat Producer by writing zero to the Heartbeat object:

0x0620	0x2b	0x17	0x10	0x00	0x00	0x00	0x00	0x00

The device should response with:

0x05a0	0x60	0x17	0x10	0x00	0x00	0x00	0x00	0x00

and now Heartbeat messages disappear from the analyzer display.

The device is still OPERATIONAL. Because the example is using simulated digital outputs we can control these now by use of PDOs. The CAN id of the first RPDO of the device is 0x0220 (0x200 + 32). This RPDO has three mapped objects, each one byte. Therefore we have to send 3 bytes with this PDO.

0x0220	0xaa	0x55	0x88					

This will show in the stdout that the device has written to all three simulated IO ports. Each byte in the date field of the CAN frame ids dedicated to one 8-bit port.

Last we can show how the CANopen device reacts on a wrongly configured RPDO. As already explained the RPDO1 is configured to receive three bytes. When we send the PDO with only two bytes.

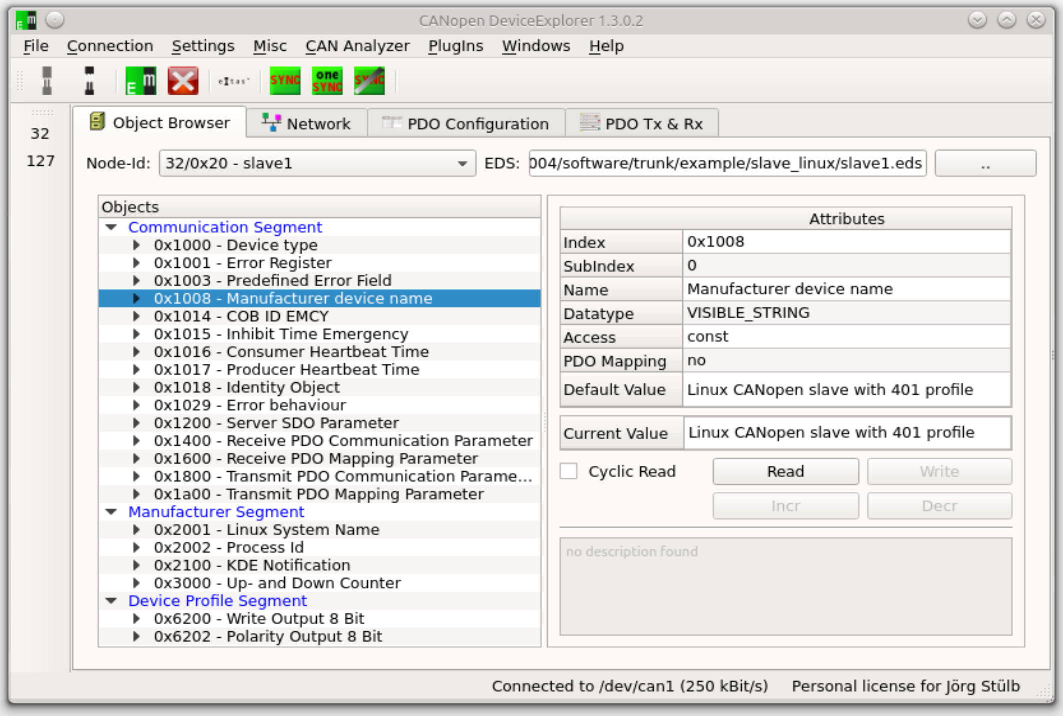
0x0220	0xaa	0x55						

we get an EMCY message sent out by the device

0x00a0	0x10	0x82	0x	0x00	0x00	0x00	0x00	0x00

The CAN id 0xa0 is composed of 0x80 + node-id, the first and second byte contain the two byte error code; always remember the little endian byte order. The value of the error code is 0x8210, which according CiA 301 is "PDO not processed due to length error". The third byte is the content of the device Error Register in Object 0x1001.

To become more familiar with CANopen it is recommended to use a CANopen configuration tool which is able to read the device EDS File. Try to read out object 0y1008, “Manufacturer device name”. This will result in the normal (segmented) SDO transfer delivering the string “Linux CANopen slave with 401 profile”.



Node-Id: 32/0x20 - slave1

EDS: 004/software/trunk/example/slave_linux/slave1.eds

Objects
- Communication Segment
  - 0x1000 - Device type
  - 0x1001 - Error Register
  - 0x1003 - Predefined Error Field
  - 0x1008 - Manufacturer device name
  - 0x1014 - COB ID EMCY
  - 0x1015 - Inhibit Time Emergency
  - 0x1016 - Consumer Heartbeat Time
  - 0x1017 - Producer Heartbeat Time
  - 0x1018 - Identity Object
  - 0x1029 - Error behaviour
  - 0x1200 - Server SDO Parameter
  - 0x1400 - Receive PDO Communication Parameter
  - 0x1600 - Receive PDO Mapping Parameter
  - 0x1800 - Transmit PDO Communication Parame...
  - 0x1a00 - Transmit PDO Mapping Parameter
- Manufacturer Segment
  - 0x2001 - Linux System Name
  - 0x2002 - Process Id
  - 0x2100 - KDE Notification
  - 0x3000 - Up- and Down Counter
- Device Profile Segment
  - 0x6200 - Write Output 8 Bit
  - 0x6202 - Polarity Output 8 Bit

Attributes	
Index	0x1008
SubIndex	0
Name	Manufacturer device name
Datatype	VISIBLE_STRING
Access	const
PDO Mapping	no
Default Value	Linux CANopen slave with 401 profile

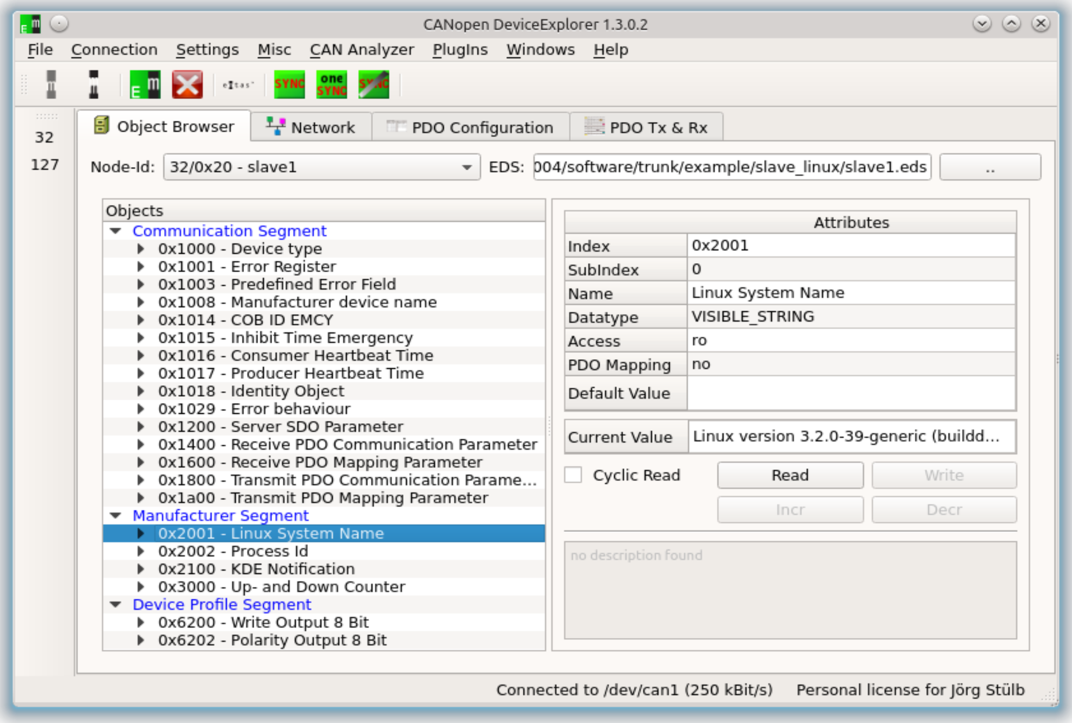
Current Value: Linux CANopen slave with 401 profile

☐ Cyclic Read

no description found

Connected to /dev/can1 (250 kBit/s) Personal license for Jörg Stülb

In the manufacturer area some more objects are accessible. Object 0x2001 again delivers a string containing the Linux version information as can be found at /proc/version.



Node-Id: 32/0x20 - slave1

EDS: 004/software/trunk/example/slave_linux/slave1.eds

Objects
- Communication Segment
  - 0x1000 - Device type
  - 0x1001 - Error Register
  - 0x1003 - Predefined Error Field
  - 0x1008 - Manufacturer device name
  - 0x1014 - COB ID EMCY
  - 0x1015 - Inhibit Time Emergency
  - 0x1016 - Consumer Heartbeat Time
  - 0x1017 - Producer Heartbeat Time
  - 0x1018 - Identity Object
  - 0x1029 - Error behaviour
  - 0x1200 - Server SDO Parameter
  - 0x1400 - Receive PDO Communication Parameter
  - 0x1600 - Receive PDO Mapping Parameter
  - 0x1800 - Transmit PDO Communication Parame...
  - 0x1a00 - Transmit PDO Mapping Parameter
- Manufacturer Segment
  - 0x2001 - Linux System Name
  - 0x2002 - Process Id
  - 0x2100 - KDE Notification
  - 0x3000 - Up- and Down Counter
- Device Profile Segment
  - 0x6200 - Write Output 8 Bit
  - 0x6202 - Polarity Output 8 Bit

Attributes	
Index	0x2001
SubIndex	0
Name	Linux System Name
Datatype	VISIBLE_STRING
Access	ro
PDO Mapping	no
Default Value	

Current Value: Linux version 3.2.0-39-generic (buildd...

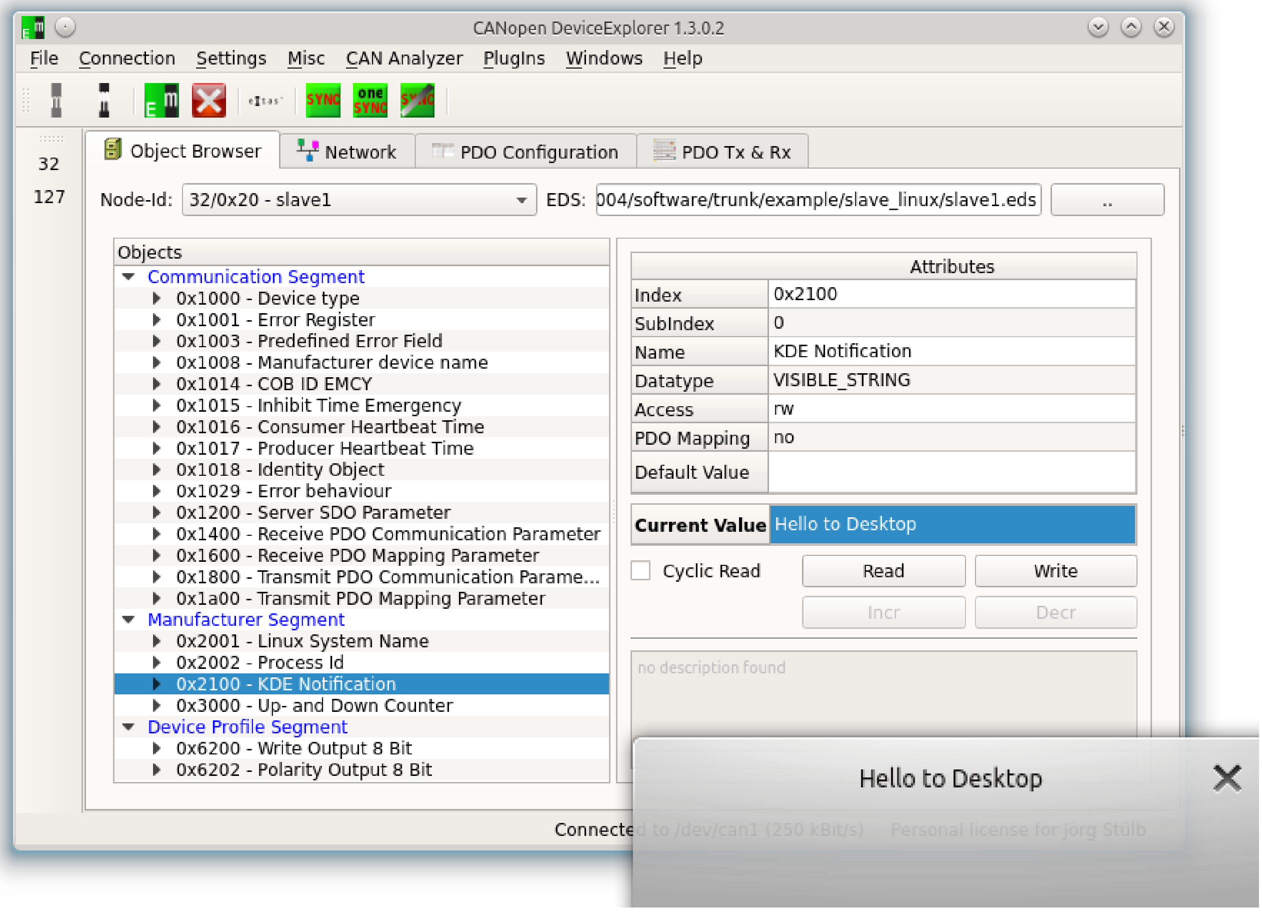
☐ Cyclic Read

no description found

Connected to /dev/can1 (250 kBit/s) Personal license for Jörg Stülb

At 0x2002 the Linux process id can be read.

Object 0x2100 is a writable string. Writing to it results on a system with KDE in a system notification on the Desktop.



Node-Id: 32/0x20 - slave1

EDS: 004/software/trunk/example/slave_linux/slave1.eds

- Communication Segment
  - 0x1000 - Device type
  - 0x1001 - Error Register
  - 0x1003 - Predefined Error Field
  - 0x1008 - Manufacturer device name
  - 0x1014 - COB ID EMCY
  - 0x1015 - Inhibit Time Emergency
  - 0x1016 - Consumer Heartbeat Time
  - 0x1017 - Producer Heartbeat Time
  - 0x1018 - Identity Object
  - 0x1029 - Error behaviour
  - 0x1200 - Server SDO Parameter
  - 0x1400 - Receive PDO Communication Parameter
  - 0x1600 - Receive PDO Mapping Parameter
  - 0x1800 - Transmit PDO Communication Parame...
  - 0x1a00 - Transmit PDO Mapping Parameter
- Manufacturer Segment
  - 0x2001 - Linux System Name
  - 0x2002 - Process Id
  - 0x2100 - KDE Notification
  - 0x3000 - Up- and Down Counter
- Device Profile Segment
  - 0x6200 - Write Output 8 Bit
  - 0x6202 - Polarity Output 8 Bit

Attributes	
Index	0x2100
SubIndex	0
Name	KDE Notification
Datatype	VISIBLE_STRING
Access	rw
PDO Mapping	no
Default Value	

Current Value Hello to Desktop

☐ Cyclic Read

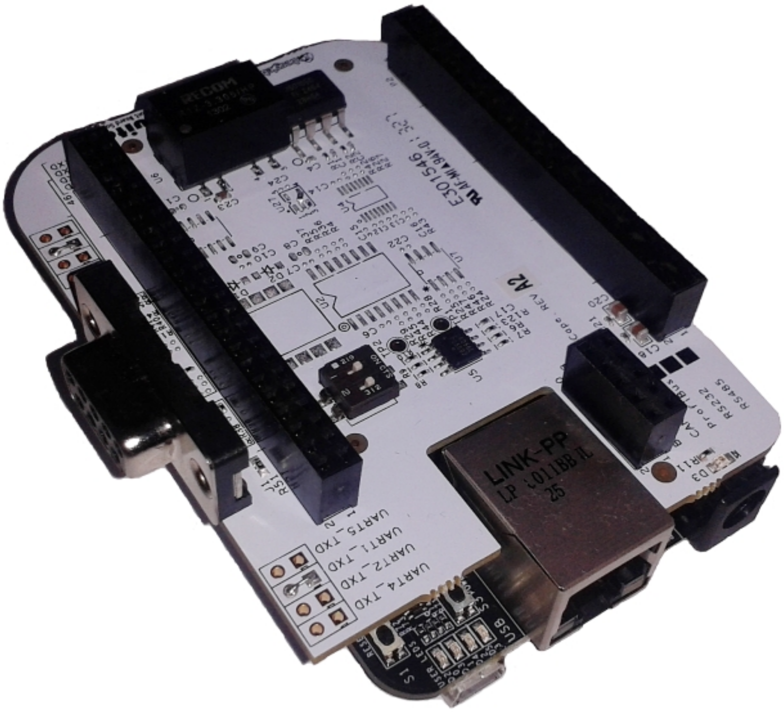
no description found

Hello to Desktop

Connected to /dev/can1 (250 kBit/s) Personal license for Jörg Stulb

4 BeagleBone Black (BBB)

The BeagleBone Black is a small Linux based single board computer. To be used with the CANopen example the CAN cape has to be added (Beagle Bone Serial Cape Rev. A2, BeagleBone CANBus Cape Rev. A2,). Before testing the CANopen example, please make sure that the used SocketCAN driver is working by testing the CAN utilities like `cansend` and `candump` (with interface can0).



4.1 Download and install the demo application

The zip archive contains the slave401 executable. Use *ftp* to transfer it to the board. The executable is a command line application. Its behavior can be controlled by command line switches. Once started information on the node state are displayed on *stdout*, for example if started via a *telnet* connection.

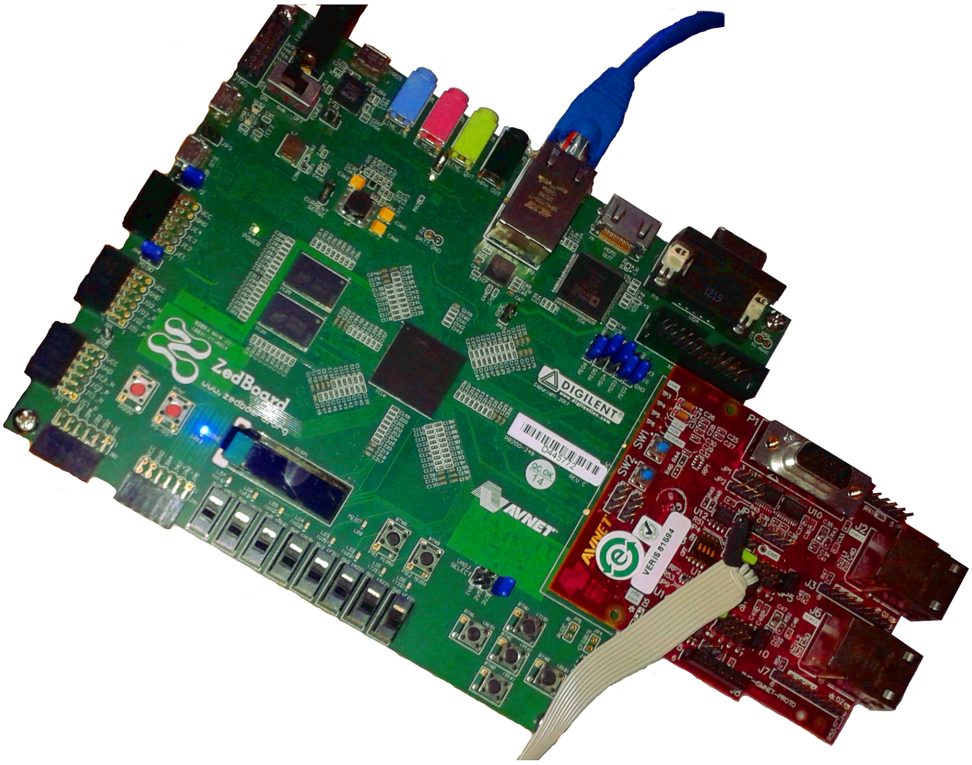
4.2 Running the tests

The CANopen example for the Beagle Bone is using the very same code and configuration as for the desktop example. Therefore the same procedure as described in3.2 can be used.

5 ZedBoard Xilinx Zynq

ZedBoard is a development platform for the Xilinx Zynq SoC. The Zynq-7000 family is based on the All Programmable SoC architecture. Zynq-7000 products incorporate a dual core ARM Cortex-A9 based Processing System (PS) and Xilinx Programmable Logic in a single device. The processing system is equipped with many interface modules like the dual channel Can controller. Because the Zedboard does not have on board CAN transceivers the FMC-ISMNET (Avnet) is used. The FMC module provides two CAN transceivers.

The Zedboard has a limited number of direct usable digital or analog IO which are not used by the CANopen example. To run the example you will need a Linux version with installed can4linux driver. Installation instructions are available at http://www.wiki.xilinx.com/CAN4Linux .



5.1 *Download and install the demo application*

The zip archive contains the slave401 executable. Use *ftp* or *scp* to transfer it to the board. The executable is a command line application. Its behavior can be controlled by command line switches. Once started information on the node state are displayed on *stdout*, for example if started via a *telnet* or *ssh* connection.

5.2 *Running the tests*

The CANopen example for the Zedboard is using the very same code and configuration as for the desktop example. Therefore the same procedure as described in3.2 can be used.