OpenXC Accessories Documentation

Release 0.1

Ford Motor Company

Nov 17, 2017

Contents

Version 0.0.1

Web http://openxcplatform.com

Documentation http://accessories.openxcplatform.com

Source http://github.com/openxc/openxc-accessories

The OpenXC Accessories are a line of hardware accessories intended to augment the Vehicle Interface (VI) and communicate with other entities. The benefit of the Accessory Platform is that all accessories share a common base (or motherboard) and new features are added by modifying or designing a new daughter card (mPCIe connector).

The base board contains an Atmel SAMA5 (Cortex-A5) running embedded Linux. All accessory functions are coded in Python. Interfaces include SD card slot, Bluetooth Classic, Bluetooth Low Energy (a.k.a Bluetooth Smart), USB OTG, and WiFi. A debug serial port is available.

The first in the line of accessories is a 3G Modem to enable sharing of vehicle data directly with the cloud, OTA updates to the Modem configuration, and still allows use of the Enabler app.

The second accessory is the V2X device. The OpenXC-V2X device can act as a modem, which connects to the VI device or a phone, and shares OpenXC data via WiFi or 802.11p (in RSU mode).

CHAPTER 1

Table of Contents

1.1 Getting Started

The OpenXC-Modem and OpenXC-V2X/RSU devices come preloaded with the kernel and firmware for operation. Before using for the first time, please charge the devices for at least 15 minutes with a micro-USB cable, or 12V wall wort. A full battery will last approximately 8-10 hours during operation.

To turn on, press the button on the side of the device once until the LED lights stay on. If there are no lights emitting from the device, ensure that the device is charged. Please note, this is a latching button that needs to be pushed in far enough to latch (a little past where the LEDs turn on).

Once the device is on, the device will automatically proceed with the auto start script, which initiates the connection and data communication with a VI and Android Device.

The following sections describe the next steps.

1.1.1 Install

Android

In order to connect with the Android device, install the *accessories* branch of the Enabler in the openxc-android project. The enabler app that works with the accessories is available here.

Be sure Bluetooth is enabled before trying to connect to the Modem. Once the Modem is running the main function, connect to the OpenXC-Modem from the Android device using Bluetooth. The password/pin code is “1234”.

Windows

The main method of configuring and setting up the modem or V2X device will be through USB from the device to a Windows PC. A program called Teraterm will be used to interface with the operating system on the device. To allow ease of use, a program called “OpenXC Modem Connect” will be used to automatically configure the connection settings.

Note: Using OpenXC Modem Connect is suggested for easier and faster access to the Modem, although you may choose to manually configure TeraTerm to connect to the modem.

Download OpenXC Modem Connect here. Detailed instructions are available here.

1.1.2 Directory Structure

The following directory structure is used.

- /root/OpenXCAccessory:

Directory Name	Description
bluez-test-script	BlueZ 5.23 test scripts (1)
openxc-python	OpenXC Python development platform. (1)
startup	Base board startup scripts (1)
common	Common Software for OpenXC-Modem/OpenXC-V2X
modem	Modem specific software
backup	Place holder for Firmware Factory Reset and current software versions. Also has backup of configuration files such as WiFi, xc.conf, boardid, and topology
etc	wpa configuration files for modem, V2X, and RSU
V2X	V2X specific Software (1)
rsu	RSU specific Software

Note: 1 - Not covered in this document

- /root/OpenXCAccessory/common

File Name	Description
xcmodem_boardid	Hidden file to specify board type: where board_type is **board _type = {** 0. {'type': 'MODEM-EVT', 'prefix': 'OpenXC-VI-MODEM'}, # OpenXC-Modem EVT 1. {'type': 'MODEM-DVT', 'prefix': 'OpenXC-VI-MODEM'}, # OpenXC-Modem DVT 2. {'type': 'V2X' , 'prefix': 'OpenXC-VI-V2X'} # OpenXC-V2X 3. {'type': 'RSU' , 'prefix': 'OpenXC-VI-V2X'} # OpenXC-V2X }
xcmodem_topology	File to specify the config mode/topology 1. Topology 1 2. Topology 2 3. Topology 3
xc_led.py	LED unit test
xc_ser.py	Serial Terminal Emulator **Usage: xcmodem_ser.py [-h] dev** where dev: Serial Device
xc_cmd.py	OpenXC-Modem application command handler and unit test
xc_app.py	OpenXC-Modem application (Mobile / PC) agent and unit test
xc_vi.py	OpenXC-Modem Vehicle Interface agent and unit test
xcmodem.conf.web	OpenXC-Modem auto start script, used during board startup
xc.conf	Local user variable options configuration file. This file is common to Modem, V2X and RSU
xc_rsu_common.py	File for RSU functions that are common to V2X and RSU
ota_upgrade.py	File for OTA upgrade functions
xc_ver.py	PpenXC-Modem version
xc_scp.pem	RSA Private Key
xc.common.py	OpenXC-Modem common functions
cleanup.py	RSU cleanup

- /root/OpenXCAccessory/modem: (applicable for OpenXC Modem Accessory only)

File Name	Description
xc.conf	Link to the xc.conf file in common directory
xcmodem.conf.web	Downloaded configuration file from remote server, if applicable
xcmodem.conf.bk	Configuration backup file which is generated during upgrading process
xcmodem.conf.cur	All options value currently in effect
trace_raw.json	Current raw VI stream snapshot in json format
trace_raw_bk.json	Back up of current raw VI stream snapshot to be processed for uploading
trace.json	Modified upload-able VI stream snapshot in json format
xcmodem_gsm.py	GSM agent and unit test
xcmodem_gsm.sh	GSM debug shell script
xcmodem_gps.py	GPS agent and unit test
xcmodem_gps.sh	GPS debug shell script

- /root/OpenXCAccessory/backup:

File Name	Description
factory	Directory to store factory released SW version info (upgrade.ver) and its upgraded package
current	Directory to store current SW version info (upgrade.ver) and its upgraded package
other	Directory to store backup of wpa_supplicant config files for Modem, V2X, RSU, and xc.conf before upgrade is performed. Boardid and topology are also backed up
previous	Directory for previous SW version during over-the-air auto upgrade, if applicable

- /root/OpenXCAccessory/v2x: (applicable for OpenXC V2X Accessory only)

File Name	Description
xc.conf	Link to the xc.conf file in common directory
xc_scp.pem	PEM key file to access AWS
xc.conf.cur	All options value currently in effect
xc_v2x.py	V2X-MODEM MD client agent and unit test

- /root/OpenXCAccessory/etc:

File Name	Description
create_symlinks.sh	Remove and replace exisiting .etc files with new files
wpa_supplicant_modem.conf	Overwrite modem configuration file whenever changed
wpa_supplicant_rsu.conf	Overwrite RSU configuration file whenever changed
wpa_supplicant_v2x.conf	Overwrite V2X configuration file whenever changed
wpa_supplicant_v2x_top2.conf	Overwrite V2X configuration file whenever changed in Topology 2

- RSU: (applicable for OpenXC V2X Accessory only)

File Name	Description
xc_rsu.py	V2X-MODEM MD client agent and unit test
rsu_fn.py	File for RSU specific functions e.g. garage

1.1.3 Scripts

Main Functions

OpenXCSoftware main functions can be performed by invoking the appropriate scripts depending on the device (Modem, V2X or RSU) as described in this section.

Modem: The Modem main function can be started by invoking xc_modem.py in /root/OpenXCAccessory/modem directory

V2X: The V2X main function can be started by invoking xc_v2x.py in /root/OpenXCAccessory/v2x directory

RSU: The RSU is a subset function of the V2X accessory. The RSU main function can be started by invoking xc_rsu.py in /root/OpenXCAccessory/rsu directory

Config Scripts

The Configuration scripts are used to setup the environment for the application. These scripts are stored in ~/OpenXCAccessory/startup directory.

Script Name	Description
openxc_init	Set the config files, Set boardid file contents, set topology, set .pem files found here.
openxc_load_config	Load /restore config files found here.
openxc_save_config	Save backup of current configuration found here.

Python Scripts

Helpful Python scripts for converting OpenXC trace files into JSON data files optimized for browsers (and Freeboard!.")

Script Name	Description	Example Usage
/openxc_json_converter.py	Takes in any trace file from the OpenXC library (examples can be downloaded from here) and converts into an array of JSON data objects. This will output a new version of the trace file named *input_trace_filename_VALIDATED.json*, which can be parsed by Freeboard datasources and widgets, and many other external APIs	`Shell $ python openxc_json_converter.py input_trace_filename.json `
/signal_extractor.py	Takes in a JSON data file (created by using /openxc_json_converter.py) and a list of signals (each prepended with '-s') that the user wishes to keep. Outputs new JSON data file with only those signals included, named *input_trace_filename_VALIDATED_STRIPPED.json*	`Shell $ python signal_extractor.py input_trace_filename_VALIDATED.json -s openxc_signal_name -s openxc_signal_name2 [...] `
/normalizer.py	Strips the input JSON data file to one data point, per signal, per second. Outputs new files named *input_trace_filename_VALIDATED_STRIPPED_NORMALIZED.json*	`Shell $ python normalizer.py input_trace_filename_VALIDATED_STRIPPED.json `

WiFi Setup

- Modem
  - The script connects the modem to one of the Access Points (APs) specified in the "wpa_supplicant" file.
  - The script opens an "OPENXC_AP" access point with 20.0.0.1 IP address for the V2X device to connect to the modem, in topology 3.
- V2X
  - The script connects the V2X device to one of the APs specified in the "wpa_supplicant" file, in topology 2.
  - The script connects to "OPENXC_AP" from modem, in topology 3.

- RSU
  - The script connects the RSU to one of the APs specified in the "wpa_supplicant" file.

Note: The scripts reset the hardware (Modem and V2X) if the required connector is not connected.

Cohda Setup

The "Cohda_setup.sh" script performs the following functions for the setting the Cohda environment and the necessary IP setup for the 802.11p based network.

- Enable Cohda HW.
- Download Firmware.
- Install llc kernel object with TCP/IP and UDP/IP support.
- Bring up Cohda interface and assign IP address.
- Create IP neighborhood for other Cohda devices (this is a pre-assigned network configuration).
  - Each Cohda device is assigned a unique 10.0.0.XX address and a unique MAC address based on the last four characters of the Bluetooth MAC address, found through a lookup table in the script.
  - All the Cohda devices in the supplied population (50 units) are added to the current device neighborhood.

1.1.4 Firmware Update

Firmware Git Pull

The best way to update firmware is with a git pull.

1. Navigate to the /root/OpenXCAccessory directory.
2. Issue a 'git pull' command.
   - Make sure the device has an Internet connection
3. Files and scripts will be updated to their latest versions from https://github.com/openxc/OpenXCAccessory.

Firmware Reset Button

1. The OpenXC-Modem and V2X Embedded SW supports a Firmware Reset Button to reset the embedded software to a known factory released version as needed.
2. Users can activate this feature by holding the button, next to the USB port, for 5 seconds (to prevent accidental triggering) once software (vi_app) is in OPERATION stage.
3. Users can also enable this feature by calling "fw_factory_reset_enable".
4. The Embedded Software will be reset to the factory released version and reboot.

Over-The-Air Auto Upgrade

1. OpenXC-Modem and V2X supports Over-The-Air Auto Upgrade

   1.1. Modem requires WiFi or GSM connection

   1.2. V2X requires WiFi connection

      1.2.1. WAN connection – upgrade file on AWS

      1.2.2. LAN via "Open_AP" – upgrade file on Modem (Modem must have the latest FW)

   1.3. During upgrade, some configurations will be backup to /root/OpenXCAccessory/backup such as wpa_supplicants, xc.conf, boardid, and toplogy

   1.4. After upgrade, user will have the option to restore configuration:

      1.4.1. All – restore all config wpa, id, topology, xc.conf

      1.4.2. Yes – option to choose, wpa configs or id, topology, xc.conf

      1.4.3. No – no restore will perform

2. Users can control this feature by calling "web_scp_sw_latest_version_url"

   The provided file from that url should contain the latest version and its associated upgraded package:

   - version
   - package

3. Modem & V2X SW will look for a newer version and perform upgrading as needed. If the upgrade fails, modem SW will perform best attempt to restore previous working version

Filesystem Upgrade

The V2X and Modem Filesystem can be upgraded using the image referenced below. The upgrade process is performed using a Linux environment.

Requirement:

1. PC with Linux OS (Ubuntu, Debian, or similar)
2. Micro SD card reader

Procedures:

1. Power on PC and boot into Linux OS
2. Download Filesystem image file

1. For V2X, use "V2X_fs_CLEAN_v2.1.1_020516.img.gz" and save to a directory
2. For Modem, use "Modem_fs_CLEAN_v2.1.1_020516.img.gz" and save to a directory
3. Open Terminal Window and type `` `sudo fdisk -l` `` and pay attention to what drive is mounted
4. Remove Micro SD from V2X and insert into card reader
5. Install card reader in Linux PC
6. In Terminal Window, type `` `sudo fdisk -l` ``

- System should detect newly insert Micro SD /dev/sdX1 and /dev/sdX2, where X is your Micro SD drive with partition 1 (sdX1) and partition 2 (sdX2)

7. Open another Terminal Window:

1. Erase all contents from Micro SDcard `` `rm -r /media/john/rootfs/*` `` or format partition 1 with ext4 and label "rootfs"
2. To copy image, type `` `sudo gunzip -c /YourDirectory/ V2X_fs_CLEAN_v2.1.1_020516.img.gz | dd of=/dev/sdX1 bs=8M` ``

   WARNING: make sure image is copied to partition 1 of Micro SD. If your system doesn't have gunzip, you will need to install with command '''apt-get -y install gzip'''

8. Safely Eject Micro SD from PC, install in device, and power it on.

Mirco SD Partition

The following procedure will guide you in how to partition a Micro SD card of any size to use for both V2X and Modem.

Requirement:

1. PC with Linux OS (Ubuntu, Debian, or similar)
2. Micro SD card reader
3. New 16GB Micro SD (recommended)

Procedures:

1. Power on PC and boot into Linux OS
2. Open Terminal Window and type `` `sudo fdisk -l` `` and pay attention to what drive is mounted
3. Remove Micro SD from V2X and insert into card reader
4. Install card reader in Linux PC
5. In Terminal Window, type `` `sudo fdisk -l` ``

- System should detect newly inserted Micro SD /dev/sdX where X is your Micro SD drive with factory partition 1 (sdX1)

6. Umount Micro SD, type `` `umount /dev/sdX1` ``
7. Start "fdisk" to partition Micro SD, type `` `sudo fdisk /dev/sdX` ``

   In command console, type the following:

   - `` `d` `` – delete partition

     1. Select correct partition to be deleted. Repeat this step if there is more than 1 partition

   - `` `n` `` – create new partition #1
   - `` `p` `` – create Primary partition #1
   - `` `1` `` – create partition #1
   - Press "Enter" – to use Default value 2048 for First Sector
   - `` `+1024M` `` – Last Sector end at 1GB
   - `` `n` `` – create new partition #2
   - `` `p` `` – create Primary partition #2
   - `` `2` `` – create partition #2
   - Press "Enter" – to use Default value for First Sector
   - Press "Enter" – to use Default value for Last Sector

- `w` – to write created partition to Micro SD

8. The newly created partition needs to be formatted, where Partition #1 use "ext4" and Partition #2 use "vfat"
   - Some Linux distributions do not come with preinstalled "dosfstools" which are required for "vfat". To install, type `apt-get -y install dosfstools`
     - This command should work for Ubuntu and Debian. Please search on how to install "dosfstools" for other Linux distros
   1. `sudo mkfs.ext4 -L rootfs /dev/sdX1` - format Partition #1 with ext4 and label "rootfs"
   2. `sudo mkfs.vfat -F 32 -n DATALOG /dev/sdX2` – format Partition #2 with vfat and label "DATALOG"
   3. Note - you may need to unmount SDcard if an error occurs when trying to format `umount /dev/sdX1`
9. Safely Eject Micro SD from PC and install to device and power it on.

1.1.5 Kernel Upgrade

In order to successfully upgrade the kernel, you will need the following two cables:

- USB-A to micro-B cable
- USB to Serial UART (FTDI TTL-232R-3V3), which can be purchased here.

Upgrade Procedure

- Power device Off.
- Remove top cover by unscrewing the 4 screws on bottom of device.
- Connect micro-B side of USB-A to micro-B cable to device.
- Connect USB-A side of cable to PC.
- Connect FTDI cable to device.
  - You will need to install the FTDI driver when connecting the cable to a PC for the first time. The FTDI driver can be downloaded from here.
  - When connecting the FTDI cable to the V2X device, make sure the Black cable on the serial connector connects to the GND pin on the V2X device. This is to ensure proper polarity.
- Connect the USB-A side of the FTDI cable to your PC and allow the FTDI driver to complete the installation.
  - Driver installation will assign a new COMx port, in addition to the USB COM port.
- Open TeraTerm and connect to the previously assigned (serial debug) COMx port with a 115200 baud rate.
  - Instructions for downloading TeraTerm can be found here.
- Power device On.
- Stop "autoboot" by pressing any key on your keyboard.
- Type "nand erase.chip" and hit Enter.
- Type "reset" and hit Enter.

- The Device Manager should have registered a new device under Ports (COM & LPT) named “AT91 USB to Serial Converter”.
  - Install or update provided driver “atm6124_cdc_signed.zip” if device did not register or install correctly.
- Install executable file “sam-ba_2.15.exe”.
- Extract KERNEL file.
  - For the V2X device, download “sama5d3_xplained-v2.1_V2X_011316.zip” to Desktop.
  - For the Modem device, download “sama5d3_xplained-v2.0_TEST_2_Modem.zip” to Desktop.
- Run “demo_linux_nandflash.bat” from extracted folder above.
  - Select “Run” on any warning popups.
- Power V2X device Off then back On after the Kernel finishes flashing to nandflash.
  - Terminal 1 will stop scrolling and Terminal 2 will automatically close.

Congratulations, you have successfully upgraded the V2X kernel.

1.2 System Overview

1.2.1 General Overview

The following section describes the high level software design for the OpenXC-Modem and V2X devices. The picture below shows the communication links between devices.

The OpenXC Embedded Software initiates connections shown in Figure 1. The devices (VI, V2X, Modem, Phone, RSU, AP and Cloud) can be configured as follows:

- Topology 1: VI + Modem + Phone + Cloud
- Topology 2: VI + V2X + RSU + Phone + Cloud
- Topology 3: VI + Modem + V2X + RSU + Phone + Cloud

1.2.2 Application Overview

The Modem, V2X and RSU devices are designed as communication sources connecting through sockets and queues.

Tasks are handled in separate threads to handle concurrent activities and exchange data safely. The threads are designed to be stoppable, using the following techniques as applicable:

- System exception to detect connection errors, or connection termination.
- Timeout exception to detect lost connection, especially in receiving/listening thread.
- External control flag to terminate execution loop.

The exchange of data from the sources to apps can be enabled or disable based on the configuration parameters described in the next section. The devices are connected through either Bluetooth, WIFi or 802.11p as shown in Figure 1.

- The Bluetooth interface uses 2 independent RFCOMM socket (Send & Recv) threads and associated data buffer queues.
- The WiFi interface uses 2 independent INET socket (Send & Recv) threads and associated data buffer queues.

- The 802.11p interface uses 2 independent UDP broadcast socket (UdpSend & UdpRecv) threads and associated data buffer queues.

1.2.3 Modem Overview

- Source: VI
  - VI through Bluetooth socket
- Applications
  - VI stream recording
  - GSM "Network Server Upload" task is handled in a separate stoppable thread
  - GPS "Acquire Current Position" task is handled in a separate stoppable thread
  - Environmental Monitor tasks (Battery level, Charger status, FW reset button ...) are handled in separate stoppable threads.
  - Mobile App Thread
  - V2X connection thread (Topology 3)

1.2.4 V2X Overview

- Sources:
  - VI through Bluetooth socket (Topology 2)
  - VI through modem over WiFI (Topology 3)
  - RSU through UDP broadcast over 802.11p
  - Self-identification announcement via UDP broadcast over 802.11p
- Applications
  - VI stream recording
  - RSU stream recording
  - Environmental Monitor tasks (Battery level, Charger status, FW reset button ...) are handled in separate stoppable threads.
  - Mobile App Thread (Topology 2)
  - VI data upload
  - RSU data upload

1.2.5 RSU Overview

- Source:
  - Garage Simulator, sends garage data through UDP broadcast over 802.11p
- Application
  - RSU data recording. Collects vehicle announcement and VI data if enabled)

1.3 Configuration

1.3.1 Configuration File

The following section describes the configuration file for the OpenXC Software.

- /root/OpenXCAccessory/common

Option Name	Unit	Default Value	Description
openxc_vi_mac	XX:XX:XX:XX:XX:XX	None	Vehicle Interface Dongle MAC
openxc_vi_enable	boolean (0 .. 1)	1/0 for MODEM/V2X	Enabling Vehicle Interface communication
openxc_md_enable	boolean (0 .. 1)	1/0 for MODEM/V2X	Enabling V2X-MD Interface communication (10)
openxc_vi_trace_snapshot_duration	seconds	10	Vehicle data stream trace recording snapshot duration
openxc_vi_trace_idle_duration	seconds	110	Idle duration between subsequent Vehicle data trace recording snapshot
openxc_vi_trace_truncate_size	bytes	0	Vehicle data trace snapshot truncate size where 0 means no truncate
openxc_vi_trace_filter_script		None	Vehicle data trace filtering executable script where the script is required to accept stdin input stream and generate stdout output (1)
openxc_vi_trace_number_of_backup	integer	0	Number of vehicle data trace will be backed up in provided micro SD card (2) where O means no back up is needed
openxc_vi_trace_backup_overwrite_enable	boolean (0 .. 1)	1	Enabling to overwrite backup files when the SD disk is full
web_scp_userid		anonymous	Remote server scp userid
v2x_lan_scp_userid		root	Remote server(Modem) scp userid for V2X in Topology 3
web_scp_pem		None Remote server SSL Enscripted Private key PEM	
web_scp_apn		apn	Remote server Access Point Name as per details provided with the SIM card contract
web_scp_config_download_enable	boolean (0 .. 1)	0	Enabling congifuration file download from remote server

Continued on next page

Table 1.1 – continued from previous page

Option Name	Unit	Default Value	Description
web_scp_config_url		ip:file	Configuration file URL on the remote server (<IP>:[<directory>/]<filename>) (3)
web_scp_vi_target_url		ip:file	Remote server target file URL in this format (<IP>:[<directory>/]<filename>) (4)
web_scp_target_overwrite_enable	boolean (0 .. 1)	1	Enabling to overwrite remote server target file (5)
web_scp_vi_trace_upload_enable	boolean (0 .. 1)	0	Enabling vehicle data records to be uploaded into remote server
web_scp_vi_trace_upload_interval	seconds	3600	Interval to upload vehicle data stream into a remote server (6)
web_scp_sw_latest_version_url		None	Auto upgrade version URL (<IP>:[<directory>/]<filename>) where None means Auto Upgrade is disable
v2x_lan_scp_sw_latest_version_url		20.0.0.1:/tmp/upgrade.ver	Auto upgrade version URL (<IP>:[<directory>/]<filename>)
fw_factory_reset_enable	boolean (0 .. 1)	1	Enabling Firmware Factory Reset Button support
power_saving_mode		Normal	Power saving profile where value is (performance / normal / saving)
led_brightness		128	LED brightness level where level is (0 .. 255) (7)
gps_log_interval	seconds	10	Interval to log GPS Acquire Current Position into /var/log/xcmodem.gps if applicable
gps_enable	boolean (0 .. 1)	1/0	for MODEM/V2X Enabling GPS module (8)

Continued on next page

Table 1.1 – continued from previous page

Option Name	Unit	Default Value	Description
gsm_enable	boolean (0 .. 1)	1/0	for MODEM/V2X Enabling GSM module (9)
openxc_v2x_trace_snapshot_duration*	seconds		RSU data stream trace recording snapshot duration for topology 3.
openxc_v2x_trace_idle_duration*	seconds		Idle duration between subsequent RSU data trace recording snapshot for topology 3
xcmodem_ip_addr	IP address	20.0.0.1	IP address for the Modem when it acts as an AP
openxc_xcV2Xrsu_trace_snapshot_duration	seconds		Duration control for RSU snapshot in V2X and RSU
openxc_xcV2Xrsu_trace_idle_duration	seconds		Interval control between RSU snapshots
web_scp_xcV2Xrsu_target_url	URL		URL for uploading RSU logs
web_scp_rxcV2Xsu_trace_upload_interval	seconds		Interval control between successive web uploads
web_scp_xcV2Xrsu_trace_upload_enable	seconds		Enable/Disable control for web upload of RSU log
openxc_xcV2Xrsu_msg_send_interval*	seconds		Control for interval between RSU identification message broadcast
chd_txpower		2 dBm	Transmit power for cohda radio
chd_radio	(‘a’..’b’)	a	Radio to be used for the Cohda module
chd_antenna	(1..3)	3	Antenna(s) to be used for radio
chd_chan_no	10 MHz channel (172, 174, 176, 180, 182, 184) 20MHz channel (175, 181) All channels SCH	184	802.11p Channel

Continued on next page

Table 1.1 – continued from previous page

Option Name	Unit	Default Value	Description
chd_modulation	MK2MCS_R12BPSK MK2MCS_R34BPSK MK2MCS_R12QPSK MK2MCS_R34QPSK MK2MCS_R12QAM16 MK2MCS_R34QAM16 MK2MCS_R23QAM64 MK2MCS_R34QAM64 MK2MCS_DEFAULT MK2MCS_TRC	MK2MCS_R12QPSK	Modulation scheme for cohda
chd_ch_update_enable	Boolean(0..1)	0	Flag to update the cohda channel parameters from the config parameters during the application run

- For optimal RSU trace recording in topology 3, trace time interval should be set as 1:2:1 ratio. Default value is 20:40:20. Where:
  - RSU device set “openxc_xcV2Xrsu_msg_send_interval = 20”
  - Modem device set “openxc_v2x_trace_snapshot_duration = 40” and “openxc_v2x_trace_idle_duration = 20”

1.3.2 Notes

1. An executable shell script like the following:

   #!/bin/bash egrep “transmission|ignition”

   will generate a trace file such as:

   {“name”:”ignition_status”,”value”:”run”,”timestamp”:1427334376.624450} {“name”:”ignition_status”,”value”:”run”,”timestamp
   {“name”:”ignition_status”,”value”:”accessory”,”timestamp”:1427334376.700860}
   {“name”:”transmission_gear_position”,”value”:”neutral”,”timestamp”:1427334376.724524}
   {“name”:”torque_at_transmission”,”value”:10.200000,”timestamp”:1427334376.734772}
   {“name”:”transmission_gear_position”,”value”:”first”,”timestamp”:1427334376.765584}
   {“name”:”ignition_status”,”value”:”run”,”timestamp”:1427334376.786151} ...

2. Raw vehicle trace snapshot will be saved as /mnt/data/trace_raw_<no>.json

   */mnt/data is mounted to the first recognized formatted partition on the inserted micro SD card

3. A unique configuration template will be created at the remote server during the device registration process, e.g: <IP>:[<directory>/]<hostname>.<filename>

   *To be used instead of provided <IP>:[<directory>/]<filename>, where <filename> is xconfig.conf by design

4. Uploading file will be named as <IP>:[<directory>/]<hostname>[.<timestamp>].<filename> at remote server where <filename> is trace.json by design

5. If overwrite flag is disabled, YYMMDDhhmmss timestamp will be added to target file name.

6. User should be aware of additional time due to trace file conversion and server connection establishment.

7. LED brightness default is 255|128|0 for performance|normal|saving of power_saving_mode respectively
8. Default value is based upon board type. This option is not valid for V2X as the V2X accessory does not support GPS.
9. Default value is based upon board type. This option is not valid for V2X as the V2X does not support GSM.
10. Default value is based upon board type. Need to be enable on both MODEM and V2X to operate V2X-Modem interface.

Power-Saving Mode Profile

To illustrate ability to support different power saving modes, OpenXC-Modem Embedded Software implements simple profiles (aka performance, normal and saving) for certain functions as shown in the following table:

1.3.3 LEDs

The Modem has 5 LED indicator lights. Battery LED has 2 colors (RED and GREEN) while the others are single color. OpenXC Modem Embedded SW controls the LEDs via gpio (/sys/class/leds/XXX).

- After power up, all LEDs except the Battery LED will blink fast.
- During software upgrades (Over-The-Air or Manufacturing Firmware Reset), all LEDs will blink slow.
- Run xcmodem.py to change LEDs according to the following table.

LED	Color Mode	Function	Keyword	State
Bat_grn_led	OFF ON FAST BLINK	VBAT < 3.55V VBAT >= 3.55V Charging	charger	NOT_CHARGE/CHARGE_DONE PRE_CHARGE/FAST_CHARGE
Bat_red_led	OFF ON FAST BLINK	VBAT > 3.65V VBAT <= 3.65V Charging	charger	NOT_CHARGE/CHARGE_DONE PRE_CHARGE/FAST_CHARGE
GSM_led	OFF ON FAST BLINK SLOW BLINK	IDLE or PPP lost GSM is ready PPP data transferring SIM not inserted	gsm_app	IDLE / LOST PENDING OPERATION PENDING
GPS_led*	OFF ON FAST BLINK SLOW BLINK	Not start GPS Unit power up Valid GPSAPC Locking for valid GPSAPC	gps_app	IDLE CONNECT OPERATION LOCKING
BT_led	OFF ON FAST BLINK SLOW BLINK	IDLE VI Dongle Connect VI Dongle Pairing VI Dongle Discovery	vi_app	IDLE / LOST OPERATION DISCOVERED ADDR_INQUIRY/ADDR_ASSIGNED
Wifi_led**	OFF ON FAST BLINK SLOW BLINK	Not Connected Connected Data Transmitting Device N/A	na	IDLE PENDING OPERATION NO WIFI DEVICE DETECTED***
80211_led	OFF FAST BLINK	Not Connected Data Transmittin	na	IDLE OPERATION

Note: .* V2X and RSU use “gps” as “wifi” led.

.** V2X and RSU use “wifi” led for 802.11p led.

.*** TI WiFi module occasionally doesn't come up during boot-up and may need manual power cycle.

Brightness Control

LED brightness is controlled by Power-saving-mode profile. However, users can overwrite the brightness level using "led_brightness" (in xcmodem.conf). The brightness level can be adjusted from 0 (dim) to 255 (bright).

1.4 Design Sources

1.4.1 Electrical

1.4.2 Mechanical

1.4.3 Assembly

1.5 License Disclosure

The OpenXC Accessories project is an open source project, and in turn depends on a few other open source projects. If you are building from source, or have downloaded a pre-compiled binary firmware, the result may contain source code covered by the following licenses:

Accessories

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of the <organization> nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL <COPYRIGHT HOLDER> BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

CHAPTER 2

License

This software depends on other open source projects, and a binary distribution may contain code covered by *other licenses*.