2020

Faro CAN Bus Module

Tony

[FARO CAN SDK REFERENCE MANUAL (LINUX)]

Version 5.0.6

Table of Contents

Revision History

Version	Date	Editor	Descriptions
V5.0.0	2020/02/26	Tony Lee	Created
V5.0.1	2020/03/18	Tony Lee	Add OBD2
V5.0.2	2020/06/05	Tony Lee	Modify OBD2 Decode
V5.0.3	2020/07/07	Vincent Cheng	Add check the amount of data function.
V5.0.4	2020/08/06	Vincent Cheng	Modify some wrong commands.
V5.0.6	2020/08/20	Vincent Cheng	Add baud rate of the COM port in the parameters of the AZ_VC_Init function.
V5.0.8	2021/09/13	Vincent Cheng	Add new initial function, AZ_VC_Init_PB

WARNING

Information in this document is subject to change without prior notice.
No part of this publication may be reproduced, transmitted, stored in a retrieval system, nor translated into any human or computer language, in any form or by any means, electronic, mechanical, magnetic, optical, chemical, manual, or otherwise, without the prior written permission of AntZer-Tech Inc.
All trademarks mentioned are proprietary of their respective owners.

1. Introduction

The Faro CAN SDK (Software Development Kit) is a software library that can be used to build the user application.
It contains many topics such as Controller Area Network (CAN), J1939, OBD-II, etc.
The Faro SDK and firmware version must be v1.01.06.20.06 (Raw CAN)/ v3.09.F4 (J1939)/v3.01.04.20.05 (J1939)/v4.01.06.20.06 (Raw CAN + OBDII + J1939 + J1708) or later.

The SDK is available as a plain „C-style SO" and can be accessed from C programming language.

2. System Requirements and Preparation

2.1 Supported Hardware:

Faro CAN bus F4 module full and half size type.

2.2 Supported OS:

Ubuntu 16.04 (kernel 4.15.0) and later.

2.3 System Prepare (Driver Installation):

Follow CP210x Linux driver “CP210x_VCP_Linux_3.13.x_Release_Notes.txt” to build and install CP210x USB-to-UART adapter driver.
Ubuntu 14.04 and later, Debian 8.9.0 has already built-in CP210x driver that means we can also use the OS default driver.

2.4 Install Necessary Building Libraries (Ubuntu):

1. Make sure your developed desktop PC or laptop connect to internet.
2. Launch terminal and enter root authority.
3. Type “sudo apt-get install build-essential cmake” to install necessary libraries and waiting for installation successfully.

2.4.1 Install Necessary Building Libraries (Debian):

1. Make sure your developed desktop PC or laptop connect to internet.
2. Launch terminal and enter root authority.
3. Type “apt-get update” to update apt latest packages.
4. Type “apt-get install build-essential cmake” to install necessary libraries and waiting for installation successfully.

2.5 Files Requirement:

Folder bin, CMakeFiles, conf, include, lib, src and files cmake_install.cmake, CMakeCache.txt, CMakeLists.txt, and Makefile.

2.6 How to build sample source code:

1. Extract “v5.0.0.tgz” (Ubuntu)(file folder name should be faro_can_sdk).
2. Switch working directory to “faro_can_sdk”.
3. Type “cmake .” on terminal window.
4. Type “make” on terminal window to build faro_can_sdk_demo.exe , faro_can_sdk_demo_j1708.exe, faro_can_sdk_demo_sensor.exe,

faro_can_sdk_receive_sample.exe sample program.

2.7 How to run sample program:

1. Type “sudo ./bin/faro_can_sdk_demo -h” on terminal window to get supporting functions list. The follow the supporting functions list description to run the function what you need.
2. The supporting functions list.

-h, --help

-s, --scan, scan available serial port

-p, --port

#1-port, ex "/dev/ttyUSB0"

-d, --deinit

#1-deinit after x seconds, -1 endless until ctrl-c

-g, --get_fw_ver, get CAN firmware version

-c, --can_config, can config

#1-port: 1 byte

#2-speed: 1 byte, 0: 1M, 1: 800K, 2: 500K, 3: 250K, 4: 200K, 5: 125K

-b, --reset_module

-m, --mode_active

#1-port: 0 or 1

#2-mode active: 0:RAW CAN, 1:OBD2, 2:J1939, 3:J1708_27byte, 4:CAN2ADR_OBD2, 5:CAN2ADR_J1939, 8:J1708_21, 9: J1708_21_OFF

-a, --set_request_time

#1-port: 0 or 1

#2-request time: 1~100ms

-f, --filter_config

-w, --filter_config_raw

#1-type: 0: ID+Mask, 1: ID List, 2: Remove, 3: Reset All

#2-port: 0, 1

#3-bank: 0 - 13

#4-mode: 0: 16 bits, 1: 32 bits

#5-filterId: 16/32 bits value in hex, ex: 0x0011/0x00112233

#6-filterId/filterMask: 16bits/32 bits in hex, ex: 0x1234/0x00112233

#7-filterMask: 16 bits in hex, ex: 0x0011

#8-filterMask: 16 bits in hex, ex: 0x0011

-x, --get_filter_config

#1 port: 0, 1

#2 bank: 0 - 13

-r, --read_write_raw

-j, --j1939_test

0 - read data

1 - write/read test

#1-port: 1 byte

#2-pgn: 4 bytes, ex: 0x18FEF100

#3-dlc: 1 byte

#4-4/8 J1939 data(0~3): 4 byte, ex: 0x01020304

#5-4/8 J1939 data(4~7): 4 byte, ex: 0x05060708

-t, --test_cnt, set counter for testing loop

-u, --get_uart_canspeed, get UART baudrate and can speed

-U, --set_uart, set UART baudrate\n"

uart baud rate - 0: 115200, 1: 230400, 2: 460800, 3: 512000, 4: 921600

-S --store_config

1: Store system configuration, 2: Factory Reset

-7, --demo_j1708

0 - read J1708 21 bytes pkt

1 - write J1708 21 bytes pkt

#1-mid: 1 byte, ex: 0x00

#2-priority: 1 byte, ex: 0x01

#3-dlc: 1 byte, ex: 0x08

#4-4/13 data: 4 byte, ex: 0x01020304

#5-4/13 data: 4 byte, ex: 0x05060708

#6-4/13 data: 4 byte, ex: 0x090A0B0C

#7-4/13 data: 4 byte, ex: 0x0D0E0F10

#8-4/13 data: 4 byte, ex: 0x11121314

--> data will be combined as 0x01020304 0x05060708 0x090A0B0C 0x0D0E0F10 0x11121314

-e, --ecompass

0 - set Accelerator Full Scale

-scale: 0: ±2, 1: ±4, 2: ±8, 3: ±16

1 - Get Accelerator Full Scale

-A, --Algorithm

0 - Clear sensor algorithm

1 - Set Fordward/Backward Algorithm Enable/Disable

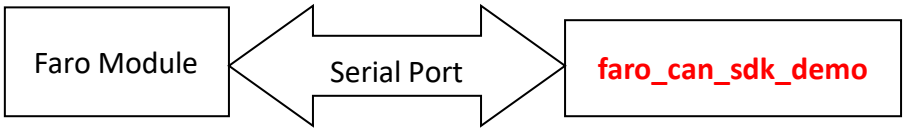
-switch: 0: Disable, 1: Enable

-k, --read_sku

-v, --version

-q, --quiet mode, set 1 to quiet mode

2.8 Faro Linux SDK File Architecture



2.9 How to receive raw CAN data:

1. Type “sudo ./bin/faro_can_sdk_receive_sample -p /dev/ttyUSB0 -r -d -1” on terminal windows and press “Enter” key to launch serial receiving thread.

2.10 How to receive J1939 data:

1. Change path to “bin” folder.
2. Type “sudo ./bin/faro_can_sdk_receive_sample -p /dev/ttyUSB0 -j -d -1” on terminal windows and press “Enter” key to launch serial receiving thread.

2.11 How to receive J1708 data (27byte):

1. Type “sudo ./bin/faro_can_sdk_demo_j1708 -p /dev/ttyUSB0 -j 0 -d -1” on terminal windows and press “Enter” key to launch serial receiving thread.

2.12 How to receive Sensor interrupt data:

1. Configure sensor interrupt (refer to
2. Type sudo ./bin/faro_can_sdk_demo_sensor -p /dev/ttyUSB0 -x 0 -d -1
3. Press combination key “ctrl+c” to exit receiving thread and it will display similar message as below.

```
Testing port = /dev/ttyUSB0, baudrate = B500000
waiting packet timeout
Testing port = /dev/ttyUSB0, baudrate = B115200
***** Adopting port = /dev/ttyUSB0, baudrate = B115200
waiting packet timeout
checking module status...
module reset complete
Calling AZ_VC_ACCCalibration successful
Calling AZ_VC_Sensor_Write, cmd = 0x53, successful
```

```
Calling AZ_VC_ConfigACCINT successful
Read packet cnt = 1
cmd = 0x93, dlc = 0x04 -->
INT_SRC = 0x01, STATUS = 0xFF, INT1_SRC = 0x60, INT2_SRC = 0x00
Read packet cnt = 2
cmd = 0x93, dlc = 0x04 -->
INT_SRC = 0x01, STATUS = 0xFF, INT1_SRC = 0x60, INT2_SRC = 0x00
Read packet cnt = 3
cmd = 0x93, dlc = 0x04 -->
INT_SRC = 0x01, STATUS = 0xFF, INT1_SRC = 0x60, INT2_SRC = 0x00
Read packet cnt = 4
cmd = 0x93, dlc = 0x04 -->
INT_SRC = 0x01, STATUS = 0xFF, INT1_SRC = 0x60, INT2_SRC = 0x00
Read packet cnt = 5
cmd = 0x93, dlc = 0x04 -->
INT_SRC = 0x01, STATUS = 0xFF, INT1_SRC = 0x60, INT2_SRC = 0x00
Read packet cnt = 6
cmd = 0x93, dlc = 0x04 -->
INT_SRC = 0x01, STATUS = 0xFF, INT1_SRC = 0x60, INT2_SRC = 0x00
Read packet cnt = 7
cmd = 0x93, dlc = 0x04 -->
INT_SRC = 0x01, STATUS = 0xFF, INT1_SRC = 0x60, INT2_SRC = 0x00
waiting packet timeout
^Ccatch SIGINT
waiting packet timeout
threads are stopped
```

3 SDK Programming API

3.1 int AZ_VC_GetCommPort(char *lpszCommPort, int *portCnt)

Include file: #include "faro_can_sdk.h"

Descriptions:

Get CAN bus module serial port name and amount

Parameters:

char * lpszCommPort – all of serial port name string and separated character is '\n', e.g. "COM3\nCOM4\n…"

int *portCnt – amount of serial port

Return value:

Success – ERROR_SUCCESS (=0), Failure – ERROR_ACCESS_DENIED and others (!=0)

Usages:

Refer to function "do_scan_port" of sample program faro_can_sdk_demo.c.

3.2 int AZ_VC_Init(char *lpszDevice, char BaudRate)

Include file: #include "faro_can_sdk.h"

Descriptions:

Serial port open, initialization & configuration

Parameters:

char *lpszDevice – serial port name string

char BaudRate – Baud rate of COM port.(0 - 115200, 1 - 230400, 2 - 460800, 3 - 921600, 4 - 921600)

Return value:

Success – ERROR_SUCCESS (=0), Failure – ERROR_ACCESS_DENIED and others (!=0)

Usages:

Refer to function "do_get_can_fw_ver" of sample program faro_can_sdk_demo.c.

3.3 int AZ_VC_Init_PB(char *lpszDevice, char BaudRate, , int bufsize)

Include file: #include "faro_can_sdk.h"

Descriptions:

Serial port open, initialization & configuration, increase ring buffer size

Parameters:

char *lpszDevice – serial port name string

char BaudRate – Baud rate of COM port.(0 - 115200, 1 - 230400, 2 - 460800, 3 - 921600, 4 - 921600)

int *bufsize – A multiple of element ring buffer size.(bufsize >= 1)

Return value:

Success – ERROR_SUCCESS (=0), Failure – ERROR_ACCESS_DENIED and others (!=0)

Usages:

Refer to function “do_get_can_fw_ver” of sample program faro_can_sdk_demo.c.

3.4 int AZ_VC_DeInit(void)

Include file: #include "faro_can_sdk.h"

Descriptions:

Serial port close & de-initialization

Parameters:

None

Return value:

Success – ERROR_SUCCESS (=0), Failure – ERROR_ACCESS_DENIED and others (!=0)

Usages:

Refer to function “deinit_wait” of sample program faro_can_sdk_demo.c.

3.5 int AZ_VC_GetFWVer(struct can_fw_ver_t *canfwver)

Include file: #include "faro_can_sdk.h"

Descriptions:

Get Faro CAN bus module firmware version

Parameters:

```
struct can_fw_ver_t
{
    uint8_t main;
    uint8_t minor;
    uint8_t type;
    uint8_t year;
    uint8_t week;
};
```

*canfwver – 3 bytes, 1st byte is major version, 2nd byte is minor version,
3rd byte is bata version, 4th byte is release year,
5th byte is release week. e.g. 1.0.f4

Return value:

Success – ERROR_SUCCESS (=0), Failure – ERROR_ACCESS_DENIED and others (!=0)

Usages:

Refer to function “do_get_can_fw_ver” of sample program faro_can_sdk_demo.c.

3.6 int AZ_VC_CAN_Config(struct can_config_t *canCFG)

Include file: #include "faro_can_sdk.h"

Descriptions:

Set CAN bus configurations

Parameters:

```
enum can_speed
{
    CAN_SPEED_1M = 0,
    CAN_SPEED_800K = 1,
    CAN_SPEED_500K = 2,
    CAN_SPEED_250K = 3,
    CAN_SPEED_200K = 4,
    CAN_SPEED_125K = 5
};
struct can_config_t
{
    uint8_t port;
    enum can_speed speed;
};
```

canCFG.port = 0 or 1

canCFG.speed = 0 ~ 5

Return value:

Success – ERROR_SUCCESS (=0), Failure – ERROR_ACCESS_DENIED and others (!=0)

Usages:

Refer to function “do_can_config” of sample program faro_can_sdk_demo.c.

3.7 int AZ_VC_CAN_Read_Amount (void)

Include file: #include "faro_can_sdk.h"

Descriptions:

Check the amount of CAN bus data received.

Parameters:

None

Return value:

The amount of CAN bus data received.

Usages:

Refer to function “do_read_write_raw” of sample program

faro_can_sdk_demo.c.

3.8 int AZ_VC_CAN_Read(struct can_message_t *candata, uint32_t *dwread)

Include file: #include "faro_can_sdk.h"

Descriptions:

The SDK support message queue method to read CAN bus data.

Parameters:

```
struct can_message_t
{
    uint8_t port;
    uint32_t id;
    uint8_t mode;    // CAN 2.0A/B (STD/Extended)
    uint8_t rtr;     // Remote request
    uint8_t dlc;
    uint8_t data[8];
};
```

*canData – total 16 bytes, 1st byte is CAN bus port 0 or 1, 2nd byte is raw CAN data ID, 3rd byte is CAN 2.0A (=0, standard) or B (=1, extended) mode, 4th byte is RTR (Remote Request) mode (=1 RTR, =0 None), 5th byte is raw CAN data length DLC, the last 8 bytes are raw CAN data.

uint32_t *dwread – actual reading bytes

Return value:

Success – ERROR_SUCCESS (=0), Failure – ERROR_ACCESS_DENIED and others (!=0)

Usages:

Refer to function “do_read_write_raw” of sample program faro_can_sdk_demo.c.

3.9 int AZ_VC_CAN_Write(struct can_message_t candata)

Include file: #include "faro_can_sdk.h"

Descriptions:

Write raw CAN data to CAN bus controller

Parameters:

```
struct can_message_t
{
    uint8_t port;
    uint32_t id;
    uint8_t mode;    // CAN 2.0A/B (STD/Extended)
    uint8_t rtr;     // Remote request
    uint8_t dlc;
    uint8_t data[8];
};
```

candata.port = 0 or 1

candata.id = 0xXXXXXXXX

candata.mode = 0 or 1

candata.rtr = 0 or 1

candata.dlc = 1 ~ 8

candata.data[0 ~ 7] = the raw CAN data would be transmitted

Return value:

Success – ERROR_SUCCESS (=0), Failure – ERROR_ACCESS_DENIED and others (!=0)

Usages:

Refer to function “do_read_write_raw” of sample program faro_can_sdk_demo.c.

3.10 int AZ_VC_CAN_Filter_Config(struct can_filter_config_t canFilterCFG)

Include file: #include "faro_can_sdk.h"

Descriptions:

Set CAN bus controller filters configuration

Parameters:

```
struct can_filter_config_t
{
    uint8_t type;    // 0 - ID+Mask, 1 - ID List, 2 - Remove, 3 - Reset All
    uint8_t port;
    uint8_t bank;
    uint8_t mode;
    uint32_t filterId;
    uint32_t filterMask;
};
```

canFilterCFG.type = 0 ~ 3

canFilterCFG.port = 0 or 1

canFilterCFG.bank = 0 ~ 14 (F1), 0 ~ 27 (F4)

canFilterCFG.mode = 0 (CAN 2.0A) or 1 (CAN 2.0B)

canFilterCFG.filterId = CAN bus controller ID filter configuration value

canFilterCFG.filterMask = CAN bus controller ID filter mask configuration value

Return value:

Success – ERROR_SUCCESS (=0), Failure – ERROR_ACCESS_DENIED and others (!=0)

Usages:

Refer to function “do_filter_config” of sample program faro_can_sdk_demo.c.

3.11 int AZ_VC_ModeActive(uint8_t mode, uint8_t mode)

Include file: #include "faro_can_sdk.h"

Descriptions:

Active and retrieve J1939, J1708, ADR_OBD2, ADR_J1939, J1708_21byte or J1708 supporting status.

Parameters:

uint8_t port – 0 or 1

uint8_t mode – 0x01: OBDII mode, 0x02: J1939 mode, 0x03: J1708 mode,
0x04: ADR_OBD2 mode, 0x05: ADR_J1939 mode
0x08: J1708_21 byte mode, 0x09: Close J1708_21 byte mode.

Return value:

Success – ERROR_SUCCESS (=0), Failure – ERROR_ACCESS_DENIED and others (!=0)

Usages:

Refer to function “do_mode_active” of sample program faro_can_sdk_demo.c.

3.12 int AZ_VC_Set_Request (uint8_t set_request_time_port, uint8_t set_request_time_time)

Include file: #include "faro_can_sdk.h"

Descriptions:

Active and retrieve J1939, J1708, ADR_OBD2, ADR_J1939, J1708_21byte or J1708 supporting status.

Parameters:

uint8_t set_request_time_port – 0 or 1

uint8_t set_request_time_time – 1 ~ 100ms
.

Return value:

Success – ERROR_SUCCESS (=0), Failure – ERROR_ACCESS_DENIED and others (!=0)

Usages:

Refer to function “do_set_request_time” of sample program faro_can_sdk_demo.c.

3.13 int AZ_VC_CAN_Filter_Get(struct can_filter_config_t *canFilterCFG)

Include file: #include "faro_can_sdk.h"

Descriptions:

Get CAN bus controller filters configuration

Parameters:

```
struct can_filter_config_t
{
    uint8_t type;    // 0 - ID+Mask, 1 - ID List, 2 - Remove, 3 - Reset All
    uint8_t port;
    uint8_t bank;
    uint8_t mode;
    uint32_t filterId;
    uint32_t filterMask;
};
```

canFilterCFG.type = 0 ~ 3

canFilterCFG.port = 0 or 1

canFilterCFG.bank = 0 ~ 14 (F1), 0 ~ 27 (F4)

canFilterCFG.mode = 0 (CAN 2.0A) or 1 (CAN 2.0B)

canFilterCFG.filterId = CAN bus controller ID filter configuration value

canFilterCFG.filterMask = CAN bus controller ID filter mask configuration value

Return value:

Success – ERROR_SUCCESS (=0), Failure – ERROR_ACCESS_DENIED and others (!=0)

Usages:

Refer to function “do_get_filter_config” of sample program faro_can_sdk_demo.c.

3.14 int AZ_VC_GetUART_CanBaudRate (struct uart_canbaudrate_t *uart_canbaudrate)

Include file: #include "faro_can_sdk.h"

Descriptions:

Get CAN uart and can speed.

Parameters:

```
struct uart_canbaudrate_t
{
    uint8_t UART; //0-115200bps,1-230400bps, 2-460800bps, 3-512000bps, 4-921600bps
```

```
    uint8_t CANPort0;
    uint8_t CANPort1;
};
```

uart_canbaudrate.UART = 0 ~ 4

uart_canbaudrate.CANPort0 = 0 ~ 5

uart_canbaudrate.CANPort1 = 0 ~ 5

Return value:

Success – ERROR_SUCCESS (=0), Failure – ERROR_ACCESS_DENIED and others (!=0)

Usages:

Refer to function “do_get_uart_canbaudrate” of sample program faro_can_sdk_demo.c.

3.15 int AZ_VC_UART_Config(uint8_t mode);

Include file: #include "faro_can_sdk.h"

Descriptions:

Set UART baud rate.

Parameters:

uint8_t mode – 0: 115200, 1: 230400, 2: 460800, 3: 512000, 4: 921600.

Return value:

Success – ERROR_SUCCESS (=0), Failure – ERROR_ACCESS_DENIED and others (!=0)

Usages:

Refer to function “do_set_uart_baudrate” of sample program faro_can_sdk_demo.c.

3.16 int AZ_VC_Save_Config (uint8_t mode);

Include file: #include "faro_can_sdk.h"

Descriptions:

Set UART baud rate.

Parameters:

uint8_t mode – 1: Store system configuration, 2: Factory Reset.

Return value:

Success – ERROR_SUCCESS (=0), Failure – ERROR_ACCESS_DENIED and others (!=0)

Usages:

Refer to function “do_set_store_config” of sample program faro_can_sdk_demo.c.

3.17 int AZ_VC_read_sku(struct module_sku_format_t module_sku);

Include file: #include "faro_can_sdk.h"

Descriptions:

Read Module SKU number.

Parameters:

uint8_t mode – 1: Store system configuration, 2: Factory Reset.

```
struct module_sku_format_t {
        uint8_t sku1;
        uint8_t sku2;
        uint8_t sku3;
        uint8_t sku4;
        uint8_t sku5;
        uint8_t sku6;
        uint8_t sku7;
        uint8_t sku8;
};
```

Return value:

Success – ERROR_SUCCESS (=0), Failure – ERROR_ACCESS_DENIED and others (!=0)

Usages:

Refer to function “do_read_sku” of sample program faro_can_sdk_demo.c

3.18 int AZ_VC_Reset_Module (void)

Include file: #include "faro_can_sdk.h"

Descriptions:

Reset Module.

Parameters:

None

Return value:

Success – ERROR_SUCCESS (=0), Failure – ERROR_ACCESS_DENIED and others (!=0)

Usages:

Refer to function “reset module” of sample program faro_can_sdk_demo.c and sample program faro_can_sdk_demo_j1708.c

*** This command is also effective when Faro is in J1708 state.**

4 OBD2 SDK Programming API

4.1 int AZ_VC_OBD2_Write(struct OBD2_send_msg_t obd2data)

Include file: #include "faro_can_sdk.h"

Descriptions:

Write OBD2 data to CAN bus controller.

Parameters:

```
struct OBD2_send_msg_t
{
    uint8_t port;
    uint32_t id;            // ID
    uint8_t PID;            // data PID
    uint8_t can_mode;       // can mode(A or B)
    uint8_t dlc;            // dlc
    uint8_t data[8];        //data
};
```

(Input)

obd2data.port = 0 or 1

obd2data.id = ex:7DF

obd2data.PID = ex:0C

obd2data.can_mode = 0: 2.0A, 1, 2.0B

obd2data.dlc = 1 ~ 8

obd2data.data[0 ~ 7] = the OBD2 data would be transmitted

Return value:

Success – ERROR_SUCCESS (=0), Failure – ERROR_ACCESS_DENIED and others (!=0)

Usages:

Refer to function “do_obd2_demo” of sample program faro_can_sdk_demo_obd2.c.

4.2 int AZ_VC_OBD2_Read_Amount (void)

Include file: #include "faro_can_sdk.h"

Descriptions:

Check the amount of OBD2 data received.

Parameters:

None

Return value:

The amount of OBD2 data received.

Usages:

Refer to function “do_obd2_demo” of sample program faro_can_sdk_demo_obd2.c.

4.3 int AZ_VC_OBD2_Read(struct OBD2_message_t *obd2data, uint32_t *dwread)

Include file: #include "faro_can_sdk.h"

Descriptions:

The SDK support message queue method to read OBD2 data from module.

Parameters:

char *lpszDevice – serial port name string

Parameters:

```
struct OBD2_message_t
{
    uint8_t port;
    uint32_t id;            // ID
    uint8_t service;        // service
    uint8_t pid;            // data PID
    uint8_t can_mode;       // can mode(A or B)
    uint8_t dlc;            // dlc
    uint8_t data[8];        //data
};
```

(output)

obd2data.port = 0 or 1

obd2data.id = ex:7DF

obd2data.service = (int value).

obd2data.pid = ex:0C

obd2data.can_mode = 0: 2.0A, 1, 2.0B

obd2data.dlc = 1 ~ 8

obd2data.data[0 ~ 7] = the OBD2 data would be transmitted

Return value:

Success – ERROR_SUCCESS (=0), Failure – ERROR_ACCESS_DENIED and others (!=0)

Usages:

Refer to function “do_obd2_demo” of sample program faro_can_sdk_demo_obd2.c.

4.4 int AZ_VC_OBD2_Read_Decode(struct OBD2_Decode_message_t *obd2data, uint8_t pid uint32_t *dwread)

Include file: #include "faro_can_sdk.h"

Descriptions:

The SDK support message queue method to read OBD2 data from module.

Parameters:

char *lpszDevice – serial port name string

Parameters:

```
struct OBD2_message_t
{
    uint8_t pid;            // data PID
    uint32_t data;          // data been decode
    uint32_t data2;         // data2 been decode
    uint8_t data_f;         // float data been decode
    uint8_t data_vi[17];    // vehicle identification number(VI)
};
```

Return value:

Success – ERROR_SUCCESS (=0), Failure – ERROR_ACCESS_DENIED and others (!=0)

Usages:

Refer to function “do_obd2_demo” of sample program faro_can_sdk_demo_obd2.c.

5 J1939 SDK Programming API

5.1 int AZ_VC_J1939_Read_Amount (void)

Include file: #include "faro_can_sdk.h"

Descriptions:

Check the amount of J1939 data received from port0(CAN1).

Parameters:

None

Return value:

The amount of J1939 data received from port0(CAN1).

Usages:

Refer to function “do_j1939_test” of sample program faro_can_sdk_demo.c.

5.2 int AZ_VC_J1939_Read(void)

Include file: #include "faro_can_sdk.h"

Descriptions:

The SDK support message queue method to read J1939 data from module can1.

Parameters:

```
struct J1939_message_t
{
    uint32_t id;
    uint8_t p;      // Priority
    uint8_t dp;     // Data Page
    uint8_t pf;     // PDU Format
    uint8_t ps;     // PDU Specific Field
    uint8_t sa;     // Source Address
    uint32_t pgn;   // Parameter Group Number
    uint8_t dlc;
    uint8_t data[8]; // Data
};
```

J1939data (output) – total 22 bytes, 1st ~ 4th byte is J1939 ID, 5th byte is J1939 data priority (0 ~ 7), 6th byte is J1939 data page (0 or 1), 7th byte is PDU format (PDU1 = 0 ~ 239, PDU3 = 240 ~ 255), 8th byte is PDU specific field, 9th byte is source address (unique address), 10th ~ 13th bytes are Parameter group number (total 3 bytes, consist of dp, pf and ps), 14th byte is raw CAN data length DLC, the last 8 bytes are raw CAN data, refer to J1939 standard SAE J1939-21 for more detail information.

UINT32 *dwread (output) – actual reading bytes

Return value:

Success – ERROR_SUCCESS (=0), Failure – ERROR_ACCESS_DENIED and others (!=0)

Usages:

Refer to function “do_j1939_test” of sample program faro_can_sdk_demo.c

5.3 int AZ_VC_J1939_Read_CAN2_Amount (void)

Include file: #include "faro_can_sdk.h"

Descriptions:

Check the amount of J1939 data received from port1(CAN2).

Parameters:

None

Return value:

The amount of J1939 data received from port1(CAN2).

Usages:

Refer to function “do_j1939_test” of sample program faro_can_sdk_demo.c.

This Command only support on faro v4.01.07.20.06, v3.01.08.20.08

5.4 int AZ_VC_J1939_Read_CAN2(struct J1939_message_t J1939data, uint32_t* dwread)

Include file: #include "faro_can_sdk.h"

Descriptions:

The SDK support message queue method to read J1939 data from module can2.

Parameters:

```
struct J1939_message_t
{
    uint32_t id;
    uint8_t p;     // Priority
    uint8_t dp;    // Data Page
    uint8_t pf;    // PDU Format
    uint8_t ps;    // PDU Specific Field
    uint8_t sa;    // Source Address
    uint32_t pgn;    // Parameter Group Number
    uint8_t dlc;
    uint8_t data[8]; // Data
};
```

J1939data (output) – total 22 bytes, 1st ~ 4th byte is J1939 ID, 5th byte is J1939 data priority (0 ~ 7), 6th byte is J1939 data page (0 or 1), 7th byte is PDU format (PDU1 = 0 ~ 239, PDU3 = 240 ~ 255), 8th byte is PDU specific field, 9th byte is source address (unique address), 10th ~ 13th bytes are Parameter group number (total 3 bytes, consist of dp, pf and ps), 14th byte is raw CAN data length DLC, the last 8 bytes are raw CAN data, refer to J1939 standard SAE J1939-21 for more detail information.

UINT32 *dwread (output) – actual reading bytes

Return value:

Success – ERROR_SUCCESS (=0), Failure – ERROR_ACCESS_DENIED and others (!=0)

Usages:

Refer to function “do_j1939_test” of sample program faro_can_sdk_demo.c.

This Command only support on faro v4.01.07.20.06, v3.01.08.20.08

5.5 int AZ_VC_J1939_Write(struct J1939_send_msg_t J1939data)

Include file: #include "faro_can_sdk.h"

Descriptions:

Write J1939 data to CAN bus controller

Parameters:

```
struct J1939_send_msg_t
{
    uint8_t port;
    uint8_t p;          // Priority
    uint8_t sa;         // Source Address
    uint32_t pgn;    // Parameter Group Number
    uint8_t dlc;
    uint8_t data[8];    // Data
};
```

(Input)

J1939data.port = 0 or 1

J1939data.p = 0 ~ 7

J1939data.sa = 0 ~ 255

J1939data.pgn = 0 ~ 131071

J1939data.dlc = 1 ~ 8

J1939data.data[0 ~ 7] = the J1939 data would be transmitted

Return value:

Success – ERROR_SUCCESS (=0), Failure – ERROR_ACCESS_DENIED and others (!=0)

Usages:

Refer to function “do_j1939_test” of sample program faro_can_sdk_demo.c.

6 J1708_21 bytes SDK Programming API

It need do AZ_VC_ModeActive(8) before calling J1708_21 bytes API (refer to 3.9 for AZ_VC_ModeActive function description).

6.1 int AZ_VC_J1708_21_Read_Amount (void)

Include file: #include "faro_can_sdk.h"

Descriptions:

Check the amount of J1708 data received on 21 bytes command.

Parameters:

None

Return value:

The amount of J1708 data received on 21 byte command.

Usages:

Refer to function “do_j1708_demo” of sample program faro_can_sdk_demo.c.

6.2 int AZ_VC_J1708_21_Read (struct J1708_21_message_t *data, uint32_t *dwread)

Include file: #include "faro_can_sdk.h"

Descriptions:

The SDK support message queue method to read J1708 data on 21 bytes command.

Parameters:

```
struct J1708_21_message_t {
    uint8_t pkt_no;      // 0 or 1
    uint8_t mid;         // 0 - 255
    uint8_t p;           // priority
    uint8_t dlc;         // 0-12
    uint8_t data[13];
};
```

data –maximum total 19 bytes,

pkt_no = 0 or 1,

mid = 0 ~ 255,

priority = 1 ~ 8,

dlc = 0 ~ 19, must match with real data[] length, data[] = buffer to write.

pkt_no is fallow by data dlc, if data dlc is bigger then 13, Then will receive 2 package data(pkt_no = 0 and 1).

UINT32 *dwread (output) – actual reading bytes

Return value:

Success –ERROR_SUCCESS (=0), Failure –ERROR_ACCESS_DENIED and others (!=0)

Usages:

Refer to function “do_j1708_demo” of sample program faro_can_sdk_demo.c.

6.3 int AZ_VC_J1708_21_Write (struct J1708_21_message_t data)

Include file: #include "faro_can_sdk.h"

Descriptions:

Write J1708 data to J1708 controller on 21 bytes command .

Parameters:

```
struct J1708_21_message_t {
    uint8_t pkt_no;      // 0 or 1
    uint8_t mid;         // 0 - 255
    uint8_t p;           // priority
    uint8_t dlc;         // 0-12
    uint8_t data[13];
};
```

data –maximum total 19 bytes,

pkt_no = 0 or 1,

mid = 0 ~ 255,

priority = 1 ~ 8,

dlc = 0 ~ 19, must match with real data[] length, data[] = buffer to write.

pkt_no is fallow by data dlc, if data dlc is bigger then 13, Then need to send 2 package for send all data(send pkt_no = 0 and 1).

Return value:

Success –ERROR_SUCCESS (=0), Failure –ERROR_ACCESS_DENIED and others (!=0)

Usages:

Refer to function “do_j1708_demo” of sample program faro_can_sdk_demo.c.

6.4 int AZ_VC_J1708_21byte_FilterConfig (uint8_t type, uint8_t mid)

Include file: #include "faro_can_sdk.h"

Descriptions:

Configure J1708 filters on 21 byte command.

Parameters:

uint8_t type = 0x01 (add filter), 0xFF (Reset all filters)

uint8_t mid = J1708 mid number for filter configuration, refer to J1708 standard for more detail information.

Return value:

Success –ERROR_SUCCESS (=0), Failure –ERROR_ACCESS_DENIED and others (!=0)

Usages:

Refer to function “do_j1708_demo” of sample program faro_can_sdk_demo.c.

6.5 int AZ_VC_J1708_21byte_GetFilterCount(uint16_t *count)

Include file: #include "faro_can_sdk.h"

Descriptions:

Get J1708 filter current total counts.

Parameters:

uint16_t count = J1708 current filter amount

Return value:

Success –ERROR_SUCCESS (=0), Failure –ERROR_ACCESS_DENIED and others (!=0)

Usages:

Refer to function “do_j1708_demo” of sample program faro_can_sdk_demo.c.

6.6 int AZ_VC_J1708_21byte_FilterGet(uint32_t *filter)

Include file: #include "faro_can_sdk.h"

Descriptions:

Get J1708 filter mid number. It must call "AZ_VC_J1708_21byte_GetFilterCount" API to get total counts first before implementing this API.

Parameters:

uint32_t filter = malloc(sizeof(uint32_t) * count); // count – filter total counts

uint32_t filter[0 ~ (count –1)] = J1708 filter mid number.

Return value:

Success –ERROR_SUCCESS (=0), Failure –ERROR_ACCESS_DENIED and others (!=0)

Usages:

Refer to function “do_j1708_demo” of sample program faro_can_sdk_demo.c.

7 J1708 SDK Programming API

It need do AZ_VC_ModeActive(3) before calling J1708 API (refer to 3.9 for AZ_VC_ModeActive function description).

7.1 int AZ_VC_J1708_Read_Amount (void)

Include file: #include "faro_can_sdk.h"

Descriptions:

Check the amount of J1708 data received.

Parameters:

None

Return value:

The amount of J1708 data received.

Usages:

Refer to function “do_j1708_test” of sample program faro_can_sdk_demo_j1708.c.

7.2 int AZ_VC_J1708_Read(struct J1708_message_t *data, uint32_t *dwread)

Include file: #include "faro_can_sdk.h"

Descriptions:

The SDK support message queue method to read J1708 data.

Parameters:

```
struct J1708_message_t {
  uint8_t mid;          // 0-255
  uint8_t p;            // priority (1-8)
  uint8_t dlc;          // 0-18
  uint8_t data[19];
};
```

*data –maximum total 22 bytes, mid = 0 ~ 255, priority = 1 ~ 8, dlc = 0 ~ 18, must match with real data[] length, data[] buffer to read.

*dwread –actual reading bytes.

Return value:

Success –ERROR_SUCCESS (=0), Failure –ERROR_ACCESS_DENIED and others (!=0)

Usages:

Refer to function “do_j1708_test” of sample program faro_can_sdk_demo_j1708.c.

7.3 int AZ_VC_J1708_Write(struct J1708_message_t data)

Include file: #include "faro_can_sdk.h"

Descriptions:

Write J1708 data to J1708 controller.

Parameters:

```
struct J1708_message_t {
   uint8_t mid; // 0-255
   uint8_t p; // priority (1-8)
   uint8_t dlc; // 0-18
   uint8_t data[19];
};
```

*data –maximum total 22 bytes, mid = 0 ~ 255, priority = 1 ~ 8, dlc = 0 ~ 18, must match with real data[] length, data[] = buffer to write.

Return value:

Success –ERROR_SUCCESS (=0), Failure –ERROR_ACCESS_DENIED and others (!=0)

Usages:

Refer to function “do_j1708_test” of sample program faro_can_sdk_demo_j1708.c.

7.4 int AZ_VC_J1708_FilterConfig(uint8_t type, uint8_t mid)

Include file: #include "faro_can_sdk.h"

Descriptions:

Configure J1708 filters.

Parameters:

type = 0 (add filter), 3 (Reset all filters)

mid = J1708 mid number for filter configuration, refer to J1708 standard for more detail information.

Return value:

Success –ERROR_SUCCESS (=0), Failure –ERROR_ACCESS_DENIED and others (!=0)

Usages:

Refer to function “do_j1708_test” of sample program faro_can_sdk_demo_j1708.c.

7.5 int AZ_VC_J1708_GetFilterCount(uint16_t *count)

Include file: #include "faro_can_sdk.h"

Descriptions:

Get J1708 filter current total counts.

Parameters:

*count = J1708 current filter amount

Return value:

Success –ERROR_SUCCESS (=0), Failure –ERROR_ACCESS_DENIED and others (!=0)

Usages:

Refer to function “do_j1708_test” of sample program faro_can_sdk_demo_j1708.c.

7.6 int AZ_VC_J1708_FilterGet(uint32_t *filter)

Include file: #include "faro_can_sdk.h"

Descriptions:

Get J1708 filter mid number. It must call "AZ_VC_J1708_GetFilterCount" API to get total counts first before implementing this API.

Parameters:

filter = malloc(sizeof(uint32_t) * count); // count – filter total counts

filter[0 ~ (count –1)] = J1708 filter mid number.

Return value:

Success –ERROR_SUCCESS (=0), Failure –ERROR_ACCESS_DENIED and others (!=0)

Usages:

Refer to function “do_j1708_test” of sample program faro_can_sdk_demo_j1708.c.

7.7 Exit J1708 mode

It must call AZ_VC_ModeActive(0) to exit J1708 mode, and then could continue using other mode, e.g. Raw CAN, J1939.

8 Sensor SDK Programming API

8.1 int AZ_VC_Sensor_Read_Amount (void)

Include file: #include "faro_can_sdk.h"

Descriptions:

Check the amount of sensor data received.

Parameters:

None

Return value:

The amount of sensor data received.

Usages:

Refer to function “do_sensor_test” of sample program faro_can_sdk_demo_sensor.c.

8.2 int AZ_VC_Sensor_Read(struct sensor_ctrl_format_t *senData,uint32_t*dwread)

Include file: #include "faro_can_sdk.h"

Descriptions:

The SDK support message queue method to read FARO sensor interrupt data.

Parameters:

```
struct sensor_ctrl_format_t
{
    byte cmd;
    byte dlc_cfg; // DLC or Configuration
    byte ctrl1;
    byte ctrl2;
    byte ctrl3;
    byte ctrl4;
    byte ctrl5;
    byte ctrl6;
    byte ctrl7;
    byte reserve[8];
};
```

*senData (output) –total 17 bytes, 1st byte is command, 2nd byte is DLC (sensor data length), 3rd ~ 9th bytes are sensor register data, Latest 8 bytes are reserved bytes (refer to appendix B).

UINT32 *dwread – actual reading bytes

Return value:

Success –ERROR_SUCCESS (=0), Failure –ERROR_ACCESS_DENIED and others (!=0)

Usages:

Refer to function “do_sensor_test” of sample program faro_can_sdk_demo_sensor.c.

8.3 int AZ_VC_Sensor_Write(struct sensor_ctrl_format_t senData)

Include file: #include "faro_can_sdk.h"

Descriptions:

Read or write ACC or Gyro sensor register data from Faro module.

Parameters:

```
struct sensor_ctrl_format_t
{
        byte cmd;
        byte dlc_cfg; // DLC or Configuration
        byte ctrl1;
        byte ctrl2;
        byte ctrl3;
        byte ctrl4;
        byte ctrl5;
        byte ctrl6;
        byte ctrl7;
        byte reserve[8];
};
```

sensorCtrl.cmd = 0x47 (Gyro) or 0x56 (ACC)

sensorCtrl.dlc_cfg = Gyro or ACC register address

sensorCtrl.ctrl1 = 0x00 or 0x01 (0 –read, 1 –write)

sensorCtrl.ctrl2 = configure register data (write) or 0x00 (read)

sensorCtrl.ctrl3 = 0x00

sensorCtrl.ctrl4 = 0x00

sensorCtrl.ctrl5 = 0x00

sensorCtrl.ctrl6 = 0x00

sensorCtrl.ctrl7 = 0x00

Return value:

Success –ERROR_SUCCESS (=0), Failure –ERROR_ACCESS_DENIED and others (!=0)

Usages:

Refer to function “do_sensor_test” of sample program faro_can_sdk_demo_sensor.c.

8.4 int AZ_VC_GetACCData(void)

Include file: #include "faro_can_sdk.h"

Descriptions:

Get ACC sensor data from Faro module.

Parameters:

None

Return value:

Success –ERROR_SUCCESS (=0), Failure –ERROR_ACCESS_DENIED and others (!=0)

Usages:

Refer to function “do_sensor_test” of sample program faro_can_sdk_demo_sensor.c.

8.5 int AZ_VC_GetACCINTData(void)

Include file: #include "faro_can_sdk.h"

Descriptions:

Get ACC sensor interrupt configuration data from Faro module.

Parameters:

None

Return value:

Success –ERROR_SUCCESS (=0), Failure –ERROR_ACCESS_DENIED and others (!=0)

Usages:

Refer to function “do_sensor_test” of sample program faro_can_sdk_demo_sensor.c.

8.6 int AZ_VC_SetACCINTData(struct sensor_ctrl_format_t ACCINTData)

Include file: #include "faro_can_sdk.h"

Descriptions:

Set ACC sensor interrupt configuration data to Faro module.

Parameters:

```
struct sensor_ctrl_format_t
{
  byte cmd;
  byte dlc_cfg; // DLC or Configuration
  byte ctrl1;
  byte ctrl2;
  byte ctrl3;
```

```
    byte ctrl4;
    byte ctrl5;
    byte ctrl6;
    byte ctrl7;
    byte reserve[8];
};
```

ACCINTData.cmd = 0x53

ACCINTData.dlc_cfg = 0x01

ACCINTData.ctrl1 = 0x10 (THS1)

ACINTData.ctrl2 = 0x00 (THS2)

ACCINTData.ctrl3 = 0x00 (DUR1)

ACCINTData.ctrl4 = 0x00 (DUR2)

ACCINTData.ctrl5 = 0x00

ACCINTData.ctrl6 = 0x00

ACCINTData.ctrl7 = 0x00

Return value:

Success –ERROR_SUCCESS (=0), Failure –ERROR_ACCESS_DENIED and others (!=0)

Usages:

Refer to function “do_sensor_test” of sample program faro_can_sdk_demo_sensor.c.

8.7 int AZ_VC_ACCCalibration(void)

Include file: #include "faro_can_sdk.h"

Descriptions:

Ask Faro module ACC sensor to do calibration.

Parameters:

None

Return value:

Success –ERROR_SUCCESS (=0), Failure –ERROR_ACCESS_DENIED and others (!=0)

Usages:

Refer to function “do_sensor_test” of sample program faro_can_sdk_demo_sensor.c.

8.8 int AZ_VC_ConfigACCINT(struct sensor_ctrl_format_t ACCINTCtrl)

Include file: #include "faro_can_sdk.h"

Descriptions:

Configure and enable ACC sensor interrupt.

Parameters:

```
struct sensor_ctrl_format_t
{
  byte cmd;
  byte dlc_cfg; // DLC or Configuration
  byte ctrl1;
  byte ctrl2;
  byte ctrl3;
  byte ctrl4;
  byte ctrl5;
  byte ctrl6;
  byte ctrl7;
  byte reserve[8];
};
```

ACCINTCtrl.cmd = 0x54

ACCINTCtrl.dlc_cfg = 0x05

ACCINTCtrl.ctrl1 = 0x70 (INT1CFG)

ACCINTCtrl.ctrl2 = 0x40 (INT1CTRL)

ACCINTCtrl.ctrl3 = 0x00 (INT2CFG)

ACCINTCtrl.ctrl4 = 0x00 (INT2CTRL)

ACCINTCtrl.ctrl5 = 0x84 (INTMODE)

ACCINTCtrl.ctrl6 = 0x00

ACCINTCtrl.ctrl7 = 0x00

Return value:

Success –ERROR_SUCCESS (=0), Failure –ERROR_ACCESS_DENIED and others (!=0)

Usages:

Refer to function “do_sensor_test” of sample program faro_can_sdk_demo_sensor.c.

8.9 int AZ_VC_GetGyroData(void)

Include file: #include "faro_can_sdk.h"

Descriptions:

Get Gyro sensor data from Faro module.

Parameters:

None

Return value:

Success –ERROR_SUCCESS (=0), Failure –ERROR_ACCESS_DENIED and others (!=0)

Usages:

Refer to function “do_sensor_test” of sample program faro_can_sdk_demo_sensor.c.

8.10 int AZ_VC_GetGyroTHS(void)

Include file: #include "faro_can_sdk.h"

Descriptions:

Get Gyro sensor threshold settings from Faro module.

Parameters:

None

Return value:

Success –ERROR_SUCCESS (=0), Failure –ERROR_ACCESS_DENIED and others (!=0)

Usages:

Refer to function “do_sensor_test” of sample program faro_can_sdk_demo_sensor.c.

8.11 int AZ_VC_GetGyroDUR(void)

Include file: #include "faro_can_sdk.h"

Descriptions:

Get Gyro sensor duration settings from Faro module.

Parameters:

None

Return value:

Success –ERROR_SUCCESS (=0), Failure –ERROR_ACCESS_DENIED and others (!=0)

Usages:

Refer to function “do_sensor_test” of sample program faro_can_sdk_demo_sensor.c.

8.12 int AZ_VC_GyroCalibration(void)

Include file: #include "faro_can_sdk.h"

Descriptions:

Ask Faro module Gyro sensor to do calibration.

Parameters:

None

Return value:

Success –ERROR_SUCCESS (=0), Failure –ERROR_ACCESS_DENIED and others (!=0)

Usages:

Refer to function “do_sensor_test” of sample program faro_can_sdk_demo_sensor.c.

8.13 int AZ_VC_SetGyroTHS(struct sensor_Gyro_format_t GyroData)

Include file: #include "faro_can_sdk.h"

Descriptions:

Set Gyro sensor threshold data to Faro module.

Parameters:

```
struct sensor_Gyro_format_t
{
  byte cmd;
  byte dlc_cfg; // DLC or Configuration
  byte dcrm;
  byte xx_H;
  byte xx_L;
  byte yy_H;
  byte yy_L;
  byte zz_H;
  byte zz_L;
  byte reserve[8];
};
```

GyroData.cmd = 0x43

GyroData.dlc_cfg = 0x07

GyroData.dcrm = 0x00 (0 –reset, 1 –decrement)

GyroData.xx_H = 0x34 (interrupt threshold on X axis high byte)

GyroData.xx_L = 0x3E (interrupt threshold on X axis low byte)

GyroData.yy_H = 0x00 (interrupt threshold on Y axis high byte)

GyroData.yy_L = 0x00 (interrupt threshold on Y axis low byte)
GyroData.zz_H = 0x00 (interrupt threshold on Z axis high byte)
GyroData.zz_L = 0x00 (interrupt threshold on Z axis low byte)

Return value:

Success –ERROR_SUCCESS (=0), Failure –ERROR_ACCESS_DENIED and others (!=0)

Usages:

Refer to function “do_sensor_test” of sample program faro_can_sdk_demo_sensor.c.

8.14 int AZ_VC_SetGyroDUR(struct sensor_GyroDUR_format_t GyroData)

Include file: #include "faro_can_sdk.h"

Descriptions:

Set Gyro sensor duration data to Faro module.

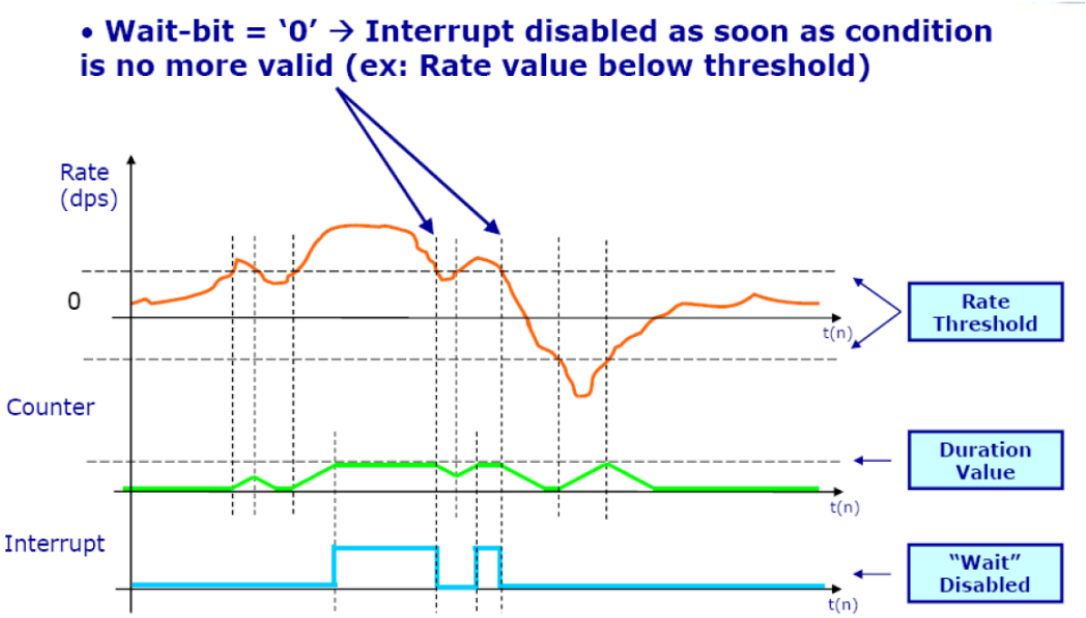
Parameters:

```
struct sensor_GyroDUR_format_t
{
  byte cmd;
  byte dlc_cfg; // DLC or Configuration
  byte wait;
  byte durData;
  byte reserve[13];
};
```

GyroData.cmd = 0x45
GyroData.dlc_cfg = 0x02
GyroData.wait = 0x00 (0 –disable, 1 –enable)
GyroData.durData = 0x7F (duration value, 0x00 ~ 0x7F)

WAIT bit has the following meaning:

Wait =’0’: the interrupt falls immediately if signal crosses the selected threshold
Wait =’1’: if signal crosses the selected threshold, the interrupt falls after a number of samples equal to the duration counter register value.



Return value:

Success –ERROR_SUCCESS (=0), Failure –ERROR_ACCESS_DENIED and others (!=0)

Usages:

Refer to function “do_sensor_test” of sample program faro_can_sdk_demo_sensor.c.

8.15 int AZ_VC_ConfigGyroINT(struct sensor_ctrl_format_t GyroCtrl)

Include file: #include "faro_can_sdk.h"

Descriptions:

Configure and enable Gyro sensor interrupt.

Parameters:

```
struct sensor_ctrl_format_t
{
  byte cmd;
  byte dlc_cfg; // DLC or Configuration
  byte ctrl1;
  byte ctrl2;
  byte ctrl3;
  byte ctrl4;
  byte ctrl5;
  byte ctrl6;
  byte ctrl7;
  byte reserve[8];
};
```

GyroCtrl.cmd = 0x49

GyroCtrl.dlc_cfg = 0x04

GyroCtrl.ctrl1 = 0x02 (INTCFG)

GyroCtrl.ctrl2 = 0x80 (INT1CTRL)

GyroCtrl.ctrl3 = 0x00 (INT2CFG)

GyroCtrl.ctrl4 = 0x00 (INT2CTRL)

GyroCtrl.ctrl5 = 0x00

GyroCtrl.ctrl6 = 0x00

GyroCtrl.ctrl7 = 0x00

Return value:

Success –ERROR_SUCCESS (=0), Failure –ERROR_ACCESS_DENIED and others (!=0)

Usages:

Refer to function “do_sensor_test” of sample program faro_can_sdk_demo_sensor.c.

8.16 int AZ_VC_clear_sensor_algorithm(void)

Include file: #include "faro_can_sdk.h"

Descriptions:

Clear Forward/backward sensor algorithm in flash.

Return value:

Success –ERROR_SUCCESS (=0), Failure –ERROR_ACCESS_DENIED and others (!=0)

Usages:

Refer to function “do_algorithm_demo” of sample program faro_can_sdk_demo.c.

8.17 int AZ_VC_Set_F_B_Algorithm (uint8_t enable)

Include file: #include "faro_can_sdk.h"

Descriptions:

Set Forward/backward Algorithm Enable/Disable.

Parameters:

uint8_t enable : 0 - Disable, 1 - Enable.

Return value:

Success –ERROR_SUCCESS (=0), Failure –ERROR_ACCESS_DENIED and others (!=0)

Usages:

Refer to function “do_algorithm_demo” of sample program faro_can_sdk_demo.c.

8.18 int AZ_VC_Get_F_B_Algorithm(struct F_B_Algorithm_t *f_b_algorithm)

Include file: #include "faro_can_sdk.h"

Descriptions:

Get Forward/backward Algorithm Enable/Disable.

Parameters:

```
struct F_B_Algorithm_t {
        uint8_t status;
        uint8_t calibration;
};
```

f_b_algorithm.status : 0 - Disable, 1 - Enable.

f_b_algorithm.calibration :

0 – No Calibration

1 - 1 Calibration

2 - 2 Calibration

3 - 3 Calibration

4 - 4 Calibration

5 - 5 Calibration

Return value:

Success –ERROR_SUCCESS (=0), Failure –ERROR_ACCESS_DENIED and others (!=0)

Usages:

Refer to function “do_algorithm_demo” of sample program faro_can_sdk_demo.c.

8.19 int AZ_VC_set_accelerator(uint8_t scale)

Include file: #include "faro_can_sdk.h"

Descriptions:

Set Accelerator Full Scale.

Parameters:

uint8_t ecompass_acc_scale : 0 ~ 3.

Return value:

Success –ERROR_SUCCESS (=0), Failure –ERROR_ACCESS_DENIED and others (!=0)

Usages:

Refer to function “do_ecompass_demo” of sample program faro_can_sdk_demo.c.

8.20 int AZ_VC_get_accelerator(struct ecompass_scale *scale)

Include file: #include "faro_can_sdk.h"

Descriptions:

Get Accelerator Full Scale.

Parameters:

```
struct ecompass_scale {
        uint8_t scale;
};
```

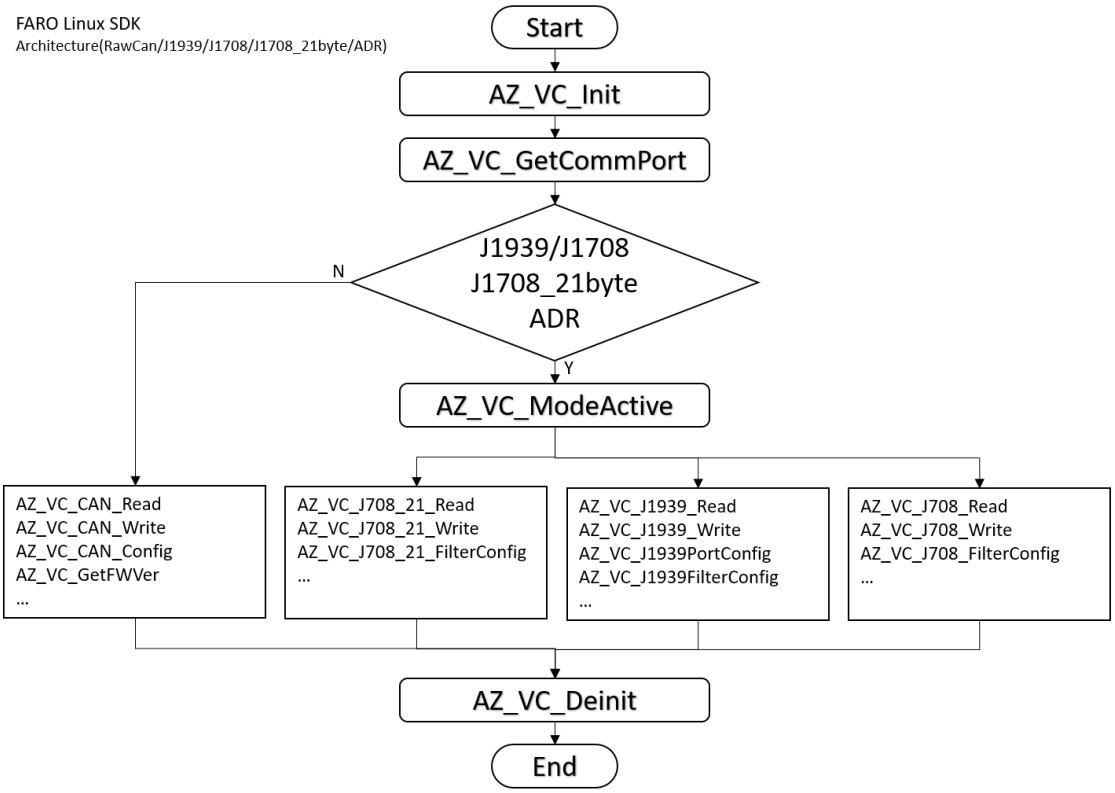
Return value:

Success –ERROR_SUCCESS (=0), Failure –ERROR_ACCESS_DENIED and others (!=0)

Usages:

Refer to function “do_ecompass_demo” of sample program faro_can_sdk_demo.c.

Appendix A : FARO SDK Architecture



Appendix B

```
struct sensor_ctrl_format_t
{
    byte cmd;
    byte dlc_cfg; // DLC or Configuration
    byte ctrl1;
    byte ctrl2;
    byte ctrl3;
    byte ctrl4;
    byte ctrl5;
    byte ctrl6;
    byte ctrl7;
    byte reserve[8];
};
```

- cmd = 0x8A (Gyro Angular Rate Data)
  dlc_cfg = 0x06
  ctrl1 = XX_L, ctrl2 = XX_H, ctrl3 = YY_L, ctrl4 = YY_H, ctrl5 = ZZ_L, ctrl6 = ZZ_H, ctrl7 = 0
- cmd = 0x8B (Gyro Threshold Data)
  dlc_cfg = 0x07
  ctrl1 = DCRM (0 – reset, 1 – decrement)
  ctrl2 = XX_H, ctrl3 = YY_L, ctrl4 = YY_H, ctrl5 = ZZ_L, ctrl6 = ZZ_H, ctrl7 = ZZ_L
- cmd = 0x8C (Gyro Duration Data)
  dlc_cfg = 0x02
  ctrl1 = Wait (0 – disable, 1 – enable)
  ctrl2 = Duration Data, ctrl3 = 0, ctrl4 = 0, ctrl5 = 0, ctrl6 = 0, ctrl7 = 0
- cmd = 0x91 (Acceleration Data, available data length = 12bit, lowest 4bit is unavailable data, negative value represents by 2's complement number)
  dlc_cfg = 0x06
  ctrl1 = XX_L, ctrl2 = XX_H, ctrl3 = YY_L, ctrl4 = YY_H, ctrl5 = ZZ_L, ctrl6 = ZZ_H, ctrl7 = 0
- cmd = 0x92 (Acceleration INT1/2 Threshold & Duration Data)
  dlc_cfg = 0x04
  ctrl1 = THS1, ctrl2 = THS2, ctrl3 = DUR1, ctrl4 = DUR2, ctrl5 = 0, ctrl6 = 0, ctrl7 = 0

- cmd = 0x93 (Response Packet during interrupt trigger)
  dlc_cfg = 0x04
  ctrl1 = Trigger source, ctrl2 = Status, ctrl3 = IG_SRC, ctrl4 = FIFO_SRC, ctrl5 = 0,
  ctrl6 = 0, ctrl7 = 0