CAN&CAN-FD Communication API User Guide

Calterah Semiconductor

Date	Version	Description	Author
2020.10.22	V1.0.3	Update CAN the transmitting data API	Xuri Liu
2020.03.12	V1.0.2	Add CAN-FD support in radar-sensor-firmware project	Xuri Liu
2019.09.27	V1.0.1	Changing the transmit order of CAN Data Byte. The default is the low byte send firstly.	Xuri Liu
2019.08.30	V1.0.0	First released version	Xuri Liu & Xudong Ran

CONTENTS

CHAPTER
ONE

Overview

This document aims to explain the data type and APIs of CAN interface communication, the definition of frame data, and the communication process in Calterah radar firmware. Please note that CAN-FD is only apply to Alps (77G) series and not apply to Rhine (60G) series.

1.1 Terms

BK: Before Kalman filtering
AK: After Kalman filtering
CAN CLI: Controller Area Network Command Line
Mgs: Message

CHAPTER
TWO

API Introduction

2.1 Data Type Definition

2.1.1 can_data_message_t

Code Path: embarc_osp\device\ip\ip_hal\inc\can_hal.h

```
typedef struct {
        /* message identifier */
        uint32_t msg_id;

        /* frame format: 0 is standard frame, 1 is extended frame */
        eCAN_FRAME_FORMAT eframe_format;

        /* frame type:     0 is data frame, 1 is remote frame */
        eCAN_FRAME_TYPE eframe_type;

        /* Length of Message Data Field , in Bytes. */
        uint32_t len;
} can_frame_params_t;

typedef struct can_data_message {
        can_frame_params_t *frame_params;
        uint8_t    data[CAN_FRAM_BUF_DATA_SIZE];
        uint32_t lock;
} can_data_message_t;
```

This data type defines the transmitting and receiving a message.

2.1.2 CAN CLI Function

Code Path: calterah\common\can_cli\can_cli.h

typedef int32_t (*can_cli_callback)(uint8_t *data, uint32_t len);

This function defines the callback function of the CAN command line.

2.2 Function Description

2.2.1 int32_t can_cli_init(void);

Code Path: calterah\common\can_cli\can_cli.c

```
/*************************Function Begin*************************
* Function Name: can_cli_init
*
* Description: It will initialize CAN hardware and create a CAN command
*              line task, then register a receiving callback function of
*              command line.
* Inputs: None
*
* Outputs: If return 0, it is OK. The others value will are abnormal.
*
*************************Function End*************************/
```

2.2.2 int32_t can_cli_register(uint32_t msg_id, can_cli_callback func);

Code Path: calterah\common\can_cli\can_cli.c

```
/*************************Function Begin*************************
* Function Name: can_cli_register
*
* Description: Register message id and a callback function of command
*              line to can_cli_cmd_list[]
* Inputs: Parameter 1 is the CAN Message ID (In some places, it might be
*         called frame ID or CAN ID.).
*         Parameter 2 is the callback function of CAN command line.
* Outputs: If return 0, it is OK. The others value will be abnormal.
*
*************************Function End*************************/
```

2.2.3 void can_send_data(uint32_t bus_id, uint32_t frame_id, uint32_t *data, uint32_t len);

Code Path: embarc_osp\device\ip\ip_hal\can_hal.c

```
/*************************Function Begin*************************
* Function Name: can_send_data
*
* Description: Transmit the CAN message to CAN network.
*
* Inputs: Parameter 1 is the CAN Device ID, or called CAN Bus ID.
*                 Parameter 2 is the message of the classical CAN frame.
* Outputs: If return 0, it is OK. The others value will be abnormal.
*
*************************Function End*************************/
```

2.2.4 int32_t can_receive_data(uint32_t dev_id, can_data_message_t *msg);

Code Path: embarc_osp\device\ip\ip_hal\can_hal.c

```
/*************************Function Begin*************************
* Function Name: can_receive_data
*
* Description: Receive the CAN message from CAN network.
*
* Inputs: Parameter 1 is a hardware CAN device id, or called CAN bus id.
*
* Outputs: Parameter 2 is a message of the classical CAN frame.
*          If return 0, it is OK. The others value will be abnormal.
*************************Function End*************************/
```


CHAPTER
THREE

There are two types of CAN message frames. They are classical CAN message frame and CAN FD message frame. The CAN message (Classical CAN message frame or CAN FD message frame) of the Alps is used for radar parameter configuration, radar state output, data input and data output. The radar sensor ID is used to identify different radars if multiple radars are connected to the CAN bus.

For example:
Code Path: calterah\common\can_cli\can_signal_interface.h

```
#define SENSOR_ID                    0
/* CAN Message ID */
#define SCAN_FRAME_ID                (0x200 + SENSOR_ID * 0x10)
#define HEAD_FRAME_ID0               (0x300 + SENSOR_ID * 0x10)
#define HEAD_FRAME_ID1               (0x301 + SENSOR_ID * 0x10)
#define BK_FRAME_ID0                 (0x400 + SENSOR_ID * 0x10)
#define BK_FRAME_ID1                 (0x401 + SENSOR_ID * 0x10)
#define BK_FRAME_ID2                 (0x402 + SENSOR_ID * 0x10)    /* Reserved */
#define AK_FRAME_ID0                 (0x500 + SENSOR_ID * 0x10)
#define AK_FRAME_ID1                 (0x501 + SENSOR_ID * 0x10)
#define AK_FRAME_ID2                 (0x502 + SENSOR_ID * 0x10)    /* Reserved */
```

Message ID is 0x200 for the radar sensor ID of 0
Message ID is 0x210 for the radar sensor ID of 1, in (0x200 + 1 * 0x10), 0x200 is the base ID.
As shown in Table 1 or Table 2 below, the Message ID is calculated given the radar sensor ID of 0. The general formula of Message ID is shown below.

Message ID = Base ID + SENSOR_ID * 0x10

3.1 Classical CAN Message Frame

Each classical CAN message ID corresponds to 8 bytes of data. The detailed CAN message frame contents please refer to the following table 1.

IN/OUT	Message ID	Message Name	Content	Section
IN	0x200	Scan	The configuration of radar scanning	3.1.1

OUT	0x300	Head data0	Output the radar track data including frame period, frame ID, BK object number	3.1.2.1
OUT	0x301	Head data1	Output the radar track data including AK object number, measurement number before pre-filter	3.1.2.1
OUT	0x400	BK data0	Output objects information before Kalman filtering, including ID number, Range, Velocity	3.1.2.2
OUT	0x401	BK data1	Output objects information before Kalman filtering, including SNR(P), Angle(A), Angle(E)	3.1.2.2
OUT	0x402	BK data2	Reserved	3.1.2.2
OUT	0x500	AK data0	Output objects information after Kalman filtering, including ID number, track status, Range, Velocity	3.1.2.3
OUT	0x501	AK data1	Output objects information after Kalman filtering, including SNR(P), Angle(A), Angle(E), Application	3.1.2.3
OUT	0x502	AK data2	Reserved	3.1.2.3

Table1: Radar Classical CAN message

3. 1.1 CAN Input Command

The Message ID of 0x200 is an input command which can configure radar scanning parameters.
One frame of data from radar includes header, BK data, and AK data. By setting the value of Message ID 0x200, the number of frame and data can be configured.

Please note that it is not supported to collect raw data through CAN bus, but the collection command can be sent through CAN with a certain frame number.
Message ID (0x200)

ID = 0x200								
	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
Byte0	7	6	5	4	3	2	1	0
	Reserved		Sample_Type			Stream_On/Off		Start/Stop
Byte1	15	14	13	12	11	10	9	8
	Number of Frame							
Byte2	23	22	21	20	19	18	17	16
	Number of Frame							
Byte3	31	30	29	28	27	26	25	24
	Reserved		0x502	0x501	0x500	0x402	0x401	0x400
Byte4	39	38	37	36	35	34	33	32
	Reserved							
Byte5	47	46	45	44	43	42	41	40
	Reserved							
Byte6	55	54	53	52	51	50	49	48
	Reserved							
Byte7	63	62	61	60	59	58	57	56
	Reserved							

Figure 1: Scan message layout (0x200)

Start / Stop: Start/Stop sensor scanning operation
No Stream: The default value, it won't do anything
Stream_On / Stream_Off: It is a switch for data acquisition
Sample Type: Type configuration of data collection
Number of Frame: Frame number of radar scan
0x400 / 0x401: To control BK data output
0x500 / 0x501: To control AK data output
0x402 / 0x502: Reserved.

Parameter	Alps Firmware Index	Width	Start Bit	End Bit	Value Range	Example	Rate	Comment
Start/Stop	NA	1	0	0	0/1		x 1	0 : Stop 1: Start
Stream_On/ Stream_Off	NA	1	1	1	0/1		x 1	00: No Stream 01 : Stream_On 10 : Stream_Off
Sample_Type	NA	3	2	4	0 to 8		x 1	000 : No Sample Type 001 : adc 010 : fft1d 011 : fft2d 100 : cfar 101 : BPM
Number of Frame	NA	16	8	23	0 to 65535		x 1	
0x400	NA	1	24	24	0/1		x 1	0: NOT output 0x400 Data 1: Output 0x400 Data
0x401	NA	1	25	25	0/1		x 1	0: NOT output 0x401 Data 1: Output 0x401 Data
0x402	NA	1	26	26	0/1		x 1	0: NOT output 0x402 Data 1: Output 0x402 Data
0x500	NA	1	27	27	0/1		x 1	0: NOT output 0x500 Data 1: Output 0x400 Data
0x501	NA	1	28	28	0/1		x 1	0: NOT output 0x501 Data 1: Output 0x401 Data
0x502	NA	1	29	29	0/1		x 1	0: NOT output 0x502 Data 1: Output 0x402 Data
Reserved	NA	34	30	63		all 0	x 1	

Figure 2: Scan message content (0x200)

3.1.2 CAN Output Data

As mentioned, one frame of data from radar contains the header, BK and AK data. The header includes the configurations of frame period, frame ID and the number of BK/AK object. BK data contains the measurements of each object including range, velocity, SNR, azimuth angle (A), elevation angle (E). AK data not only contains the measured values of each object, including range, velocity, SNR, azimuth angle (A), elevation angle (E), but also adds track level and radar application function information.

3.1.2.1 Header

Message ID (0x300)

ID = 0x300

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
Byte0	7	6	5	4	3	2	1	0
	Frame Period							
Byte1	15	14	13	12	11	10	9	8
	Reserved						Frame Period	
Byte2	23	22	21	20	19	18	17	16
	Frame ID #							
Byte3	31	30	29	28	27	26	25	24
	Frame ID #							
Byte4	39	38	37	36	35	34	33	32
	Frame ID #							
Byte5	47	46	45	44	43	42	41	40
	Frame ID #							
Byte6	55	54	53	52	51	50	49	48
	cdi_number (BK ID #)							
Byte7	63	62	61	60	59	58	57	56
	Reserved						cdi_number	

Figure 3: Head data0 message layout (0x300)

Frame Period: Radar Scanning Period
Frame ID: ID of Frame
cdi_number: The number of the detected object before Kalman filter and after pre-filter

Parameter	Alps Firmware Index	Width	Start Bit	End Bit	Value Range	Example	Rate	Comment
Frame Period	track->output_hdr.frame_int	10	0	9	0ms to 1023ms	63ms - 0x3F - 0b 00 0011 1111 127ms - 0x7F - 0b 00 0111 1111 1023ms -0x3FF - 0b 11 1111 1111	x 0.001	
Reserved	NA	6	10	15		all 0	x 1	
Frame ID #	track->output_hdr.frame_id	32	16	47	0 to 4,294,967,295	15 - 0x00F(15) -0b 0000 0000 0000 0000 0000 0000 0000 1111 858 - 0x35A(858) - 0b 0000 0000	x 1	
cdi_number	track->cdi_pkg.cdi_number (BK Object #)	10	48	57	0 to 1023	15 - 0x00F(15) -0b 00 0000 1111 440 - 0x1B8(440) - 0b 01 1011 1000 511 - 0x1FF(511) - 0b 01 1111 1111 875 - 0x36B(875) - 0b 11 0110 1011	x 1	
Reserved	NA	6	58	63		all 0	x 1	

Figure 4: Head data0 message content (0x300)

Message ID (0x301)

ID = 0x301

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
Byte0	7	6	5	4	3	2	1	0
	track_output_number (AK Object ID #)							
Byte1	15	14	13	12	11	10	9	8
	Reserved						track_output_number	
Byte2	23	22	21	20	19	18	17	16
	raw_number							
Byte3	31	30	29	28	27	26	25	24
	Reserved						raw_number	
Byte4	39	38	37	36	35	34	33	32
	Reserved							
Byte5	47	46	45	44	43	42	41	40
	Reserved							
Byte6	55	54	53	52	51	50	49	48
	Reserved							
Byte7	63	62	61	60	59	58	57	56
	Reserved							

Figure 5: Head data1 message layout (0x301)

track_output_number: The number of the detected object after Kalman filter
raw_number: The number of the detected object before pre-filter

Parameter	Alps Firmware Index	Width	Start Bit	End Bit	Value Range	Example	Rate	Comment
track_output_number	track->output_hdr.track_output_number (AK Object #)	10	0	9	0 to 1023	15 - 0x00F(15) -0b 00 0000 1111 440 - 0x1B8(440) - 0b 01 1011 1000 511 - 0x1FF(511) - 0b 01 1111 1111 875 - 0x36B(875) - 0b 11 0110 1011	x 1	
Reserved	NA	6	10	15		all 0	x 1	
raw_number	track->cdi_pkg.raw_number	10	16	25	0 to 1023	15 - 0x00F(15) -0b 00 0000 1111 440 - 0x1B8(440) - 0b 01 1011 1000 511 - 0x1FF(511) - 0b 01 1111 1111 875 - 0x36B(875) - 0b 11 0110 1011	x 1	
Reserved	NA	38	26	63		all 0	x 1	

Figure 6: Head data1 message content (0x301)

3.1.2.2 BK Data

Message ID (0x400)

ID = 0x400

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
Byte0	7	6	5	4	3	2	1	0
	BK ID Number							
Byte1	15	14	13	12	11	10	9	8
	Reserved						BK ID Number	
Byte2	23	22	21	20	19	18	17	16
	Range(R)							
Byte3	31	30	29	28	27	26	25	24
	Range(R)							
Byte4	39	38	37	36	35	34	33	32
	Range(R)							
Byte5	47	46	45	44	43	42	41	40
	Range(R)							
Byte6	55	54	53	52	51	50	49	48
	Velocity(V)							
Byte7	63	62	61	60	59	58	57	56
	Reserved	Velocity(V)						

Figure 7: BK data0 message layout (0x400)

BK ID Number: Object ID before Kalman filtering.
Range: Radial distance between the radar and the object.
Velocity: Relative velocity between the radar and the object, also called Doppler velocity.

Parameter	Alps Firmware Index	Width	Start Bit	End Bit	Value Range
BK ID Number	i	10	0	9	0 to 1023
Reserved	NA	6	10	15	
Range	track->cdi_pkg.cdi[i].raw_z.rng	32	16	47	Depend on Firmware
Velocity	track->cdi_pkg.cdi[i].raw_z.vel	15	48	62	-163.84m/s to +163.83m/s
Reserved	NA	1	63	63	

Figure 8.0: BK data0 message content (0x400)

Parameter	Example	Rate	Comment
BK ID Number	15 - 0x00F(15) -0b 00 0000 1111 440 - 0x1B8(440) - 0b 01 1011 1000 511 - 0x1FF(511) - 0b 01 1111 1111 875 - 0x36B(875) - 0b 11 0110 1011	x 1	
Reserved	all 0	x 1	
Range	Depend on Firmware	x1	Get the real Range(float) value from firmware
Velocity	150.03m/s : 0x3A9B(15003) +0x4000 = 0x7A9B - 0b 111 1010 1001 1011 163.83m/s : 0x3FFF(16383) + 0x4000 = 0x7FFF - 0b 111 1111 1111 1111 0m/s : 0x0000(0) + 0x4000 = 0x4000 - 0b 100 0000 0000 0000 -163.84m/s : 0x4000 - 0x4000(16384) = 0x0000 - 0b 0000 0000 0000 0000	x 0.01	For a positive Velocity value(am/s), send (V = a*100+0x4000) to PC. For a negative Velocity value(-am/s), send (V = 0x4000 - la|*100) to PC. For 0m/s, send (V = 0x4000) to PC. GUI on PC calculate real velocity value by : (V -0x4000) * 0.01
Reserved	all 0	x 1	

Figure 8.1: BK data0 example (0x400)

Message ID (0x401)

ID = 0x401

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
Byte0	7	6	5	4	3	2	1	0
	SNR(P)							
Byte1	15	14	13	12	11	10	9	8
	Reserved	SNR(P)						
Byte2	23	22	21	20	19	18	17	16
	Angle(A)							
Byte3	31	30	29	28	27	26	25	24
	Reserved	Angle(A)						
Byte4	39	38	37	36	35	34	33	32
	Angle(E)							
Byte5	47	46	45	44	43	42	41	40
	Reserved	Angle(E)						
Byte6	55	54	53	52	51	50	49	48
	Reserved							
Byte7	63	62	61	60	59	58	57	56
	Reserved							

Figure 9: BK data1 message layout (0x401)

SNR (P): Signal-to-noise ratio
Angle (A): Azimuth angle
Angle (E): Elevation angle

Parameter	Alps Firmware Index	Width	Start Bit	End Bit	Value Range
SNR(P)	10*log10f(tmpS/tmpN)	15	0	14	-163.84dB to +163.83dB
Reserved	NA	1	15	15	
Angle(A)	track->cdi_pkg.cdi[i].raw_z.ang	15	16	30	-163.84deg to +163.83deg
Reserved	NA	1	31	31	
Angle(E)	track->cdi_pkg.cdi[i].raw_z.ang_elv	15	32	46	-163.84deg to +163.83deg
Reserved	NA	17	47	63	

Figure 10.0: BK data1 message content (0x401)

Parameter	Example	Rate	Comment
SNR(P)	28.63dB : 0x0B2F(2863) + 0x4000 = 0x4B2F -0b 100 1011 0010 1111 51.18dB : 0x13FE(5118) + 0x4000 = 0x53FE - 0b 101 0011 1111 1110 0dB : 0x0000(0) + 0x4000 = 0x4000 - 0b 100 0000 0000 0000 -87.68dB : 0x4000 - 0x2240(8768) = 0x1DC0 - 0b 001 1101 1100 0000	x 0.01	For a positive SNR value(adB), send (SNR = a*100+0x4000) to PC. For a negative SNR value(-adB), send (SNR = 0x4000 - lal*100) to PC. For 0dB, send (SNR = 0x4000) to PC. GUI on PC calculate real SNR value by : (SNR -0x4000) * 0.01
Reserved	all 0	x 1	
Angle(A)	41.8deg : 0x1A2(418) + 0x4000 = 0x41A2 - 0b 0100 0001 1010 0010 102.3deg : 0x3FF(1023) + 0x4000 = 0x43FF - 0b 0100 0011 1111 1111 0deg : 0x0000(0) + 0x4000 = 0x4000 - 0b 100 0000 0000 0000 -102.47deg : 0x4000 - 0x2807(10247) = 0x17F9 - 0b 0001 0111 1111 1001 -15.88deg : 0x4000 - 0x634(1588) = 0x39CC - 0b 0011 1001 1100 1100	x 0.01	For a positive Angle(A) value(adeg), send (Deg = a*100+0x4000) to PC. For a negative Angle(A) value(-adeg), send (Deg = 0x4000 - lal*100) to PC. For 0deg, send (Deg = 0x4000) to PC. GUI on PC calculate real Angle(A) value by : (Deg -0x4000) * 0.01
Reserved	all 0	x 1	
Angle(E)	41.8deg : 0x1A2(418) + 0x4000 = 0x41A2 - 0b 0100 0001 1010 0010 102.3deg : 0x3FF(1023) + 0x4000 = 0x43FF - 0b 0100 0011 1111 1111 0deg : 0x0000(0) + 0x4000 = 0x4000 - 0b 100 0000 0000 0000 -102.47deg : 0x4000 - 0x2807(10247) = 0x17F9 - 0b 0001 0111 1111 1001 -15.88deg : 0x4000 - 0x634(1588) = 0x39CC - 0b 0011 1001 1100 1100	x 0.01	For a positive Angle(E) value(adeg), send (Deg = a*100+0x4000) to PC. For a negative Angle(E) value(-adeg), send (Deg = 0x4000 - lal*100) to PC. For 0deg, send (Deg = 0x4000) to PC. GUI on PC calculate real Angle(E) value by : (Deg -0x4000) * 0.01
Reserved	all 0	x 1	

Figure 10.1: BK data1 example (0x401)

Message ID (0x402)

ID = 0x402

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
Byte0	7	6	5	4	3	2	1	0
	Reserved							
Byte1	15	14	13	12	11	10	9	8
	Reserved							
Byte2	23	22	21	20	19	18	17	16
	Reserved							
Byte3	31	30	29	28	27	26	25	24
	Reserved							
Byte4	39	38	37	36	35	34	33	32
	Reserved							
Byte5	47	46	45	44	43	42	41	40
	Reserved							
Byte6	55	54	53	52	51	50	49	48
	Reserved							
Byte7	63	62	61	60	59	58	57	56
	Reserved							

Figure 11: BK data2 message content (0x402)

Parameter	Alps Firmware Index	Width	Start Bit	End Bit	Value Range
Reserved	NA	64	0	63	

Figure 12.0: BK data2 message content (0x402)

Parameter	Example	Rate	Comment
Reserved	all 0	x 1	

Figure 12.1: BK data2 example (0x402)

3.1.2.3 AK Data

Message ID (0x500)

ID = 0x500

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
Byte0	7	6	5	4	3	2	1	0
	AK Object ID #							
Byte1	15	14	13	12	11	10	9	8
	Reserved				Track Status		BK ID Number	
Byte2	23	22	21	20	19	18	17	16
	Range(R)							
Byte3	31	30	29	28	27	26	25	24
	Range(R)							
Byte4	39	38	37	36	35	34	33	32
	Range(R)							
Byte5	47	46	45	44	43	42	41	40
	Range(R)							
Byte6	55	54	53	52	51	50	49	48
	Velocity(V)							
Byte7	63	62	61	60	59	58	57	56
	Reserved	Velocity(V)						

Figure 13: AK data0 message layout (0x500)

AK Object ID: Object ID after Kalman filtering.

Track Status: Status of tracking. 0 means that the object has not been tracked. 1 means that the object

is moving towards the radar. 2 means that the object is moving away from the radar.
Range: Radial distance between the radar and the object.
Velocity: Relative velocity between the radar and the object, also called Doppler velocity.

Parameter	Alps Firmware Index	Width	Start Bit	End Bit	Value Range
AK Object ID #	i	10	0	9	0 to 1023
Track Status	track->output_obj.track_level	2	10	11	0 to 3
Reserved	NA	4	12	15	
Range	track->output_obj.rng	32	16	47	Depend on Firmware
Velocity	track->output_obj.vel	15	48	62	-163.84m/s to +163.83m/s
Reserved	NA	1	63	63	

Figure 14.0: AK data0 message content (0x500)

Parameter	Example	Rate	Comment
AK Object ID #	15 - 0x00F(15) -0b 00 0000 1111 440 - 0x1B8(440) - 0b 01 1011 1000 511 - 0x1FF(511) - 0b 01 1111 1111 875 - 0x36B(875) - 0b 11 0110 1011	x 1	
Track Status	1 - 0x1 -0b 01 3 - 0x11 -0b 11	x 1	
Reserved	all 0	x 1	
Range	Depend on Firmware	x1	Get the real Range(float) value from firmware
Velocity	150.03m/s : 0x3A9B(15003) +0x4000 = 0x7A9B - 0b 111 1010 1001 1011 163.83m/s : 0x3FFF(16383) + 0x4000 = 0x7FFF - 0b 111 1111 1111 1111 0m/s : 0x0000(0) + 0x4000 = 0x4000 - 0b 100 0000 0000 0000 -163.84m/s : 0x4000 - 0x4000(16384) = 0x0000 - 0b 0000 0000 0000 0000 -126.85m/s : 0x4000 - 0x318D(12685) = 0xE73 - 0b 0000 1110 0111 0011	x 0.01	For a positive Velocity value(am/s), send (V = a*100+0x4000) to PC. For a negative Velocity value(-am/s), send (V = 0x4000 - |a|*100) to PC. For 0m/s, send (V = 0x4000) to PC. GUI on PC calculate real velocity value by : (V -0x4000) * 0.01
Reserved	all 0	x 1	

Figure 14.1: AK data0 example (0x500)

Message ID (0x501)

ID = 0x501

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
Byte0	7	6	5	4	3	2	1	0
	SNR(P)							
Byte1	15	14	13	12	11	10	9	8
	Reserved	SNR(P)						
Byte2	23	22	21	20	19	18	17	16
	Angle(A)							
Byte3	31	30	29	28	27	26	25	24
	Reserved	Angle(A)						
Byte4	39	38	37	36	35	34	33	32
	Angle(E)							
Byte5	47	46	45	44	43	42	41	40
	Reserved	Angle(E)						
Byte6	55	54	53	52	51	50	49	48
	Application							
Byte7	63	62	61	60	59	58	57	56
	Application							

Figure 15: AK data1 message layout (0x501)

SNR (P): Signal-to-noise ratio
Angle (A): Azimuth angle
Angle (E): Elevation angle
Application: Functional switches for radar applications, such as BSD/Sensor door and etc.

Parameter	Alps Firmware Index	Width	Start Bit	End Bit	Value Range
SNR(P)	track->output_obj.SNR	15	0	14	-163.84dB to +163.83dB
Reserved	NA	1	15	15	
Angle(A)	track->output_obj.ang	15	16	30	-163.84deg to +163.83deg
Reserved	NA	1	31	31	
Angle(E)	track->output_obj.ang_elv	15	32	46	-163.84deg to +163.83deg
Reserved	NA	11	47	47	
Application	Not defined yet	16	48	63	

Figure 16.0: AK data1 message content (0x501)

Parameter	Example	Rate	Comment
SNR(P)	28.63dB : 0x0B2F(2863) + 0x4000 = 0x4B2F -0b 100 1011 0010 1111 51.18dB : 0x13FE(5118) + 0x4000 = 0x53FE - 0b 101 0011 1111 1110 0dB : 0x0000(0) + 0x4000 = 0x4000 - 0b 100 0000 0000 0000 -87.68dB : 0x4000 - 0x2240(8768) = 0x1DC0 - 0b 001 1101 1100 0000	x 0.01	For a positive SNR value(adB), send (SNR = a*100+0x4000) to PC. For a negative SNR value(-adB), send (SNR = 0x4000 - |a|*100) to PC. For 0dB, send (SNR = 0x4000) to PC. GUI on PC calculate real SNR value by : (SNR -0x4000) * 0.01
Reserved	all 0	x 1	
Angle(A)	41.8deg : 0x1A2(418) + 0x4000 = 0x41A2 - 0b 0100 0001 1010 0010 102.3deg : 0x3FF(1023) + 0x4000 = 0x43FF - 0b 0100 0011 1111 1111 0deg : 0x0000(0) + 0x4000 = 0x4000 - 0b 100 0000 0000 0000 -102.47deg : 0x4000 - 0x2807(10247) = 0x17F9 - 0b 0001 0111 1111 1001 -15.88deg : 0x4000 - 0x634(1588) = 0x39CC - 0b 0011 1001 1100 1100	x 0.01	For a positive Angle(A) value(adeg), send (Deg = a*100+0x4000) to PC. For a negative Angle(A) value(-adeg), send (Deg = 0x4000 - |a|*100) to PC. For 0deg, send (Deg = 0x4000) to PC. GUI on PC calculate real Angle(A) value by : (Deg -0x4000) * 0.01
Reserved	all 0	x 1	
Angle(E)	41.8deg : 0x1A2(418) + 0x4000 = 0x41A2 - 0b 0100 0001 1010 0010 102.3deg : 0x3FF(1023) + 0x4000 = 0x43FF - 0b 0100 0011 1111 1111 0deg : 0x0000(0) + 0x4000 = 0x4000 - 0b 100 0000 0000 0000 -102.47deg : 0x4000 - 0x2807(10247) = 0x17F9 - 0b 0001 0111 1111 1001 -15.88deg : 0x4000 - 0x634(1588) = 0x39CC - 0b 0011 1001 1100 1100	x 0.01	For a positive Angle(E) value(adeg), send (Deg = a*100+0x4000) to PC. For a negative Angle(E) value(-adeg), send (Deg = 0x4000 - |a|*100) to PC. For 0deg, send (Deg = 0x4000) to PC. GUI on PC calculate real Angle(E) value by : (Deg -0x4000) * 0.01
Reserved	all 0	x 1	
Application	Currently all 0 - 0b 0 0000 0000 0000 0000		Reserved 16bits for Application function Use

Figure 16.1: AK data1 example (0x501)

Message ID (0x502)

ID = 0x502

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
Byte0	7	6	5	4	3	2	1	0
	Reserved							
Byte1	15	14	13	12	11	10	9	8
	Reserved							
Byte2	23	22	21	20	19	18	17	16
	Reserved							
Byte3	31	30	29	28	27	26	25	24
	Reserved							
Byte4	39	38	37	36	35	34	33	32
	Reserved							
Byte5	47	46	45	44	43	42	41	40
	Reserved							
Byte6	55	54	53	52	51	50	49	48
	Reserved							
Byte7	63	62	61	60	59	58	57	56
	Reserved							

Figure 17: AK data2 message layout (0x502)

Parameter	Alps Firmware Index	Width	Start Bit	End Bit	Value Range
Reserved	NA	64	0	63	

Figure 18.0: AK data2 message content (0x502)

Parameter	Example	Rate	Comment
Reserved	all 0	x 1	

Figure 18.1: AK data2 example (0x502)

3.2 CAN FD Message Frame

Each CAN FD message ID corresponds to 16 bytes of data. Therefore, the CAN FD message frame can hold more data.
The detailed CAN FD message frame contents please refer to the following table 2.

IN/OUT	Message ID	Message Name	Content	Section
IN	0x200	Scan	The configuration of radar scanning	3.1.1
OUT	0x300	Head data0	Output the radar track data including frame period, frame ID, BK object number, AK object number, measurement number before pre-filter.	3.1.2.1
OUT	0x301	Head data1	Reserved	3.1.2.1
OUT	0x400	BK data0	Output objects information before Kalman filtering, including ID number, Range, Velocity, SNR(P), Angle(A), Angle(E)	3.1.2.2
OUT	0x401	BK data1	Reserved	3.1.2.2
OUT	0x402	BK data2	Reserved	3.2.1.2
OUT	0x500	AK data0	Output objects information after Kalman filtering, including ID number, track status, Range, Velocity, SNR(P), Angle(A), Angle(E), Application	3.1.2.3
OUT	0x501	AK data1	Reserved	3.1.2.3
OUT	0x502	AK data2	Reserved	3.1.2.3

Table2: Radar CAN FD Messages

3.2.1 CAN FD Input Command

The Message ID of 0x200 is also an input command of CAN FD. The Message ID contents is the same as Classical CAN. Only its data length is 16 bytes. So the command define please refer to the Classical CAN.

3.2.2 CAN FD Output Data

The radar data output by CAN FD is the same as the classical CAN output. It's just that the frame space of CAN FD has increased, and it is now 16 bytes per frame. So it will transmit radar information with fewer CAN message IDs. Now the HEAD data tracked by the radar only needs a CAN message ID 0x300, BK data tracked by the radar only needs a CAN message ID 0x400, and AK data tracked by the radar only needs a CAN message ID 0x500. The contents of the CAN message tracked by the radar and the encoding method remain unchanged. Same as classical CAN frame. So the output data define please refer to the Classical CAN.

CHAPTER
FOUR

CAN Communication Process

CAN communication can be divided into the following six stages.

4.1 Hardware Initialization

The initialization of CAN controller and transceiver, including setting the Base ID of CAN register, enabling CAN clock, initializing communication baud rate to 500Kbps, and etc.

4.2 Software Initialization

- Invoke CAN installation function *can_install_ops(&can_r2p0_0)*
- Initialize CAN interrupt service routine with *int_handler_install(…)*
- Enable interrupt service *int_enable(…)*
- Create CAN ISR queue with *queue_can_isr = xQueueCreate(…)*
  - queue_can_isr is used to synchronize with can_cli_task
- Create CAN Command Line task, *xTaskCreate(can_cli_handle, can_cli_task…)*
  - can_cli_task deals with the query and parse of CAN command
- Register the callback fucntion of receiving CAN message,
  can_indication_register(0, can_cli_rx_indication)
  - can_cli_rx_indication saves received CAN message to rx_msg_buffer[]
- Register the callback function of parsing CAN message, *can_cli_register(…)*
  - CAN ID will be stored in can_cli_cmd_list[]. can_scan_signal parses CAN message, and start radar scanning operation.

Please note that if no message received on the CAN bus, CAN ISR queue will force CAN command line into the block state. Only when message received on the CAN bus will queue cancel the block state of CAN command line, otherwise, CAN command line will stay in block state.

4.3 Receiving Message on the CAN Bus

When message available on the CAN bus, ISR (interrupt service routine) will be triggered. In ISR, messages on CAN bus are saved to rx_msg_buffer[] by _can_frame_read function, and the block state will be cancelled.

4.4 Parsing CAN Message

CAN command line task, can_cli_task, parses CAN messages in rx_msg_buffer[]. If CAN Message ID is in can_cli_cmd_list[], the callback function, can_scan_signal, for parsing CAN message will be invoked to start functional operation of radar.

4.5 Executing Radar Algorithm and Send out on CAN Bus

Baseband Task executes radar algorithm, calculating the tracking result, and sending the message on CAN bus through can_transmit_data().

4.6 Parsing Received Message on CAN Bus

The CAN communication flow diagram is shown in Figure 19.

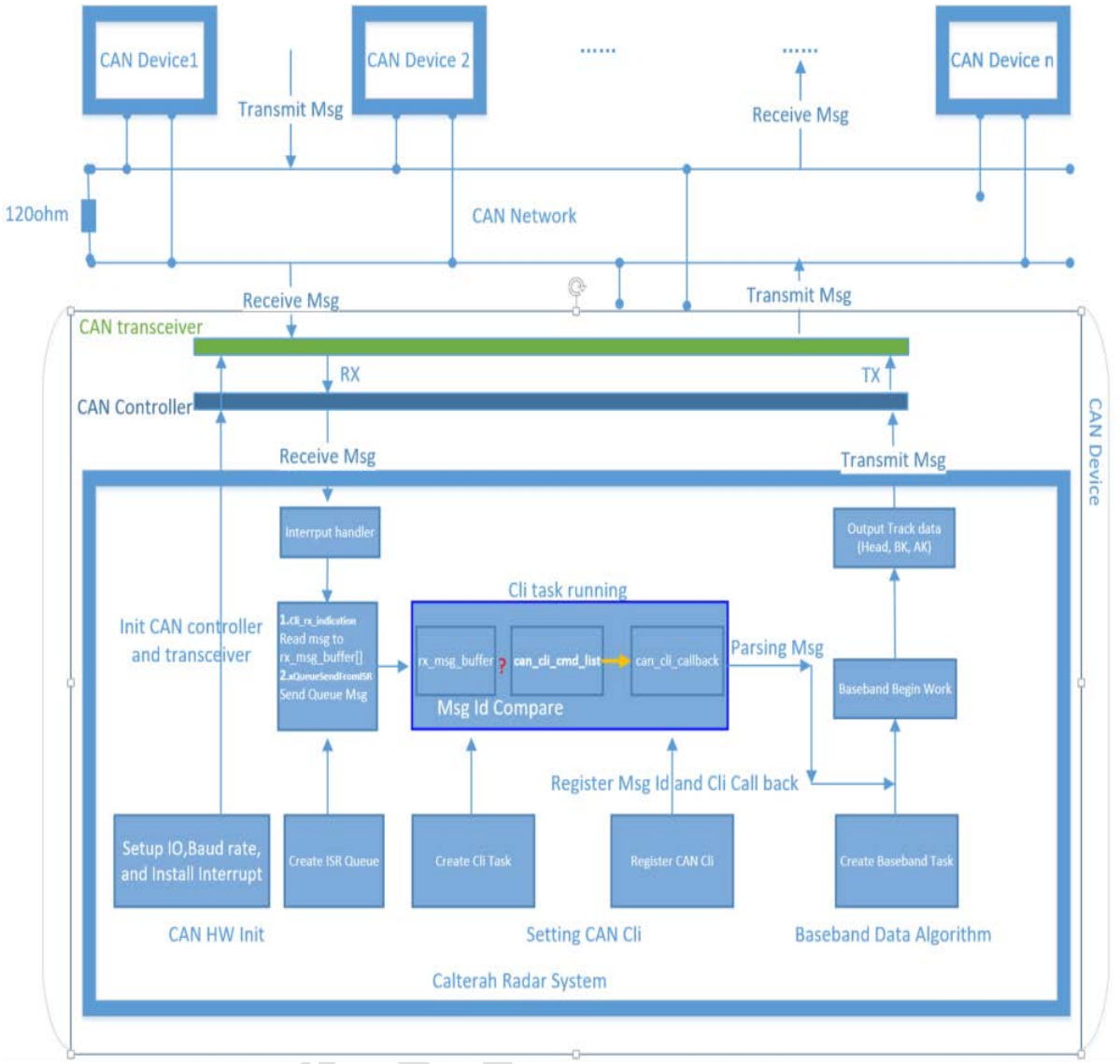

Figure 19 CAN communication flow

CHAPTER
FIVE

CAN Communication Verification

5.1 Classic CAN Communication Verification

Hardware and software requirements:
<1> Calterah Radar Development Platform Alps
<2> Classical CAN Analysis Tool: CANalyst-II (No support CAN FD)
<3> Personal computer
<4> Classical CAN analysis software: USB_CAN Tool V2.1.2 (No support CAN FD)
<5> UART debugging tool SecureCRT 8.0

5.1.1 Classical CAN Device Connection

Connect Micro-USB port on Alps with USB Type-A on a personal computer. The USB connection on Alps is used for power supply as well as sending logs through UART interface.

Connect CAN Analysis Tool to PC through a USB port.
Connect CAN 1 on CAN Analysis Tool (CANalyst-II) with CAN 0 bus on Alps.

Figure 20 Classical CAN Device Connection

5.1.2 Classical CAN tool Software Installation

<1> Install CAN analysis software USB_CAN Tool
<2> Install UART debugging tool SecureCRT 8.0

5.1.3 Classic CAN Communication Debugging

<1> Launch USB-CAN Tool V2.12. Select the proper device in Device menu, and start device in Operation - Start.

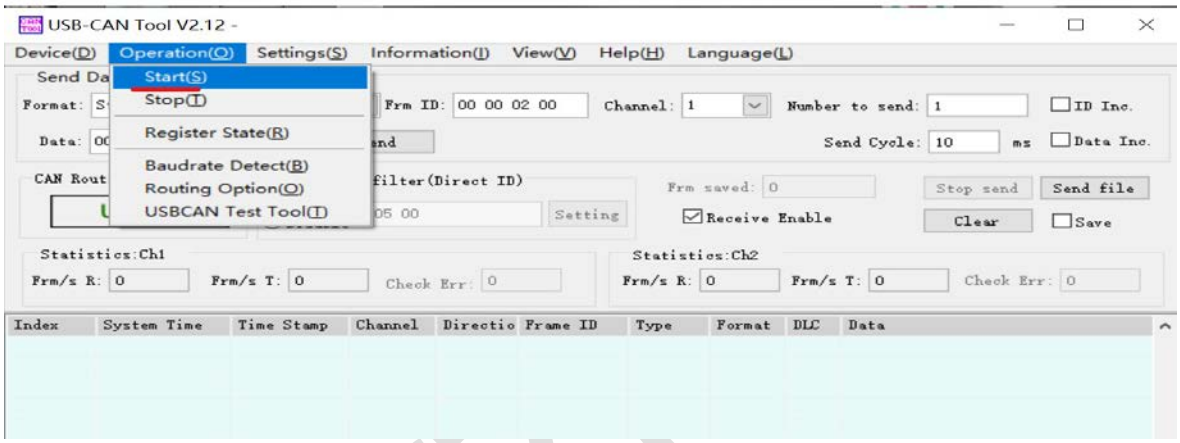

Figure 21.0 USB-CAN Tool

<2> In the popup window, select Ch1, set the baud rate to 500k bps (the same baud rate in firmware), and press OK.

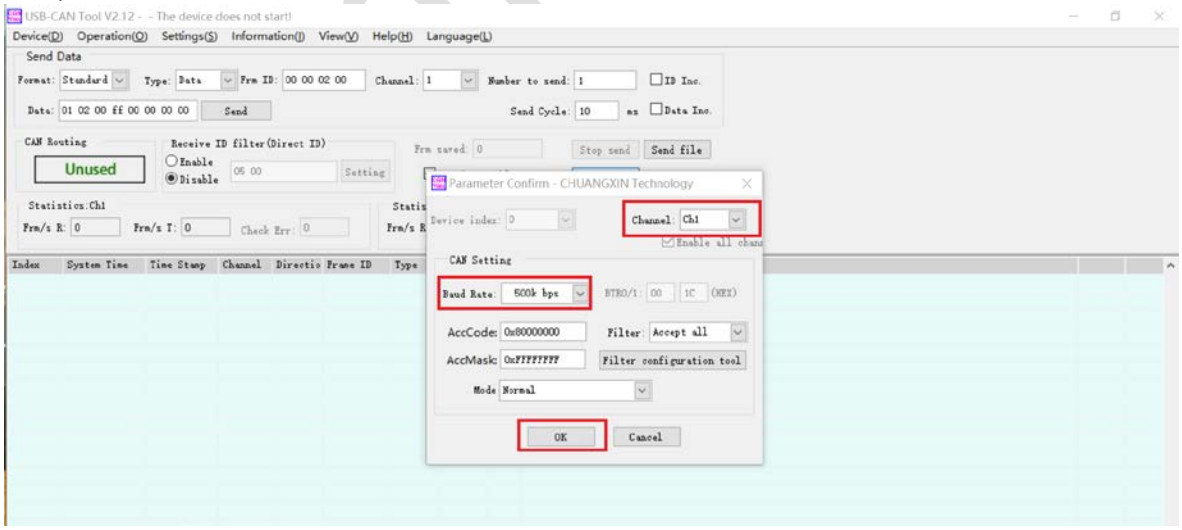

Figure 21.1 USB-CAN Tool

<3> A popup window shows Device Start. Press OK for the next step.

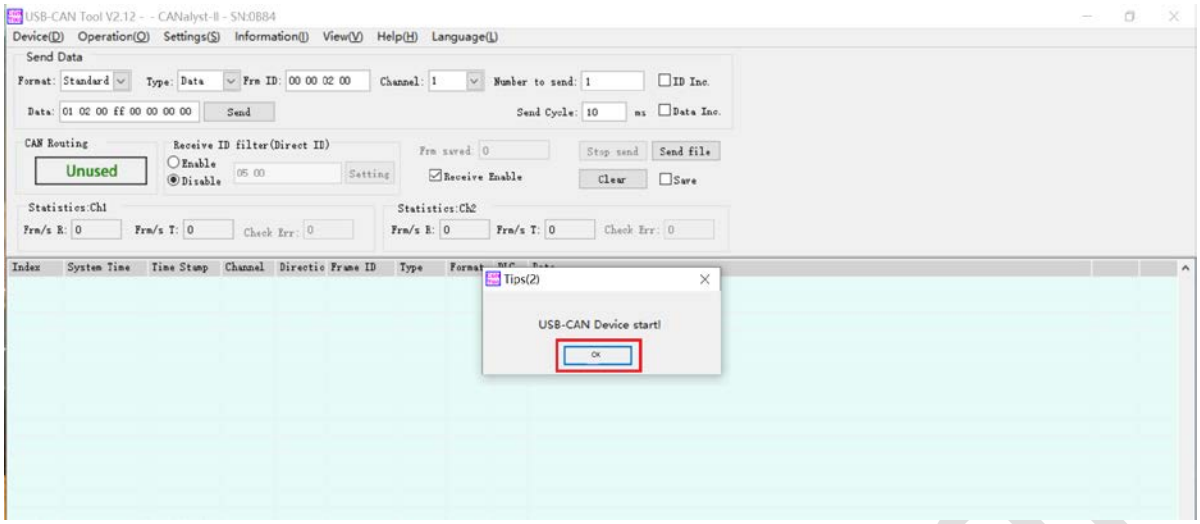

Figure 21.2 USB-CAN Tool

<4> CAN Message Configuration
Configure CAN message as follow.
Format: Standard
Type: Data
Frame ID: 0x200
Channel: 1
Data: 01 02 00 ff 00 00 00 00
(The least 1 byte [01] represents start scanning. The second and third bytes [0002] represents the number of the frame for scanning is two. The highest one-byte [FF] represents outputting all tracking data including header, BK and AK.)

In the data input box sent by CAN, the first byte ~ the eighth byte of the data field in the CAN message is in order from left to right. For example, "01 02 00 ff 00 00 00 00" is filled in the input box, and the 64-bit unsigned number composed from low to high is 0x00000000ff000201. In the list of CAN messages, first low and then high are displayed as "01 02 00 ff 00 00 00 00" (default display mode).

Press **Send** and the tracking data will be displayed in the output window.

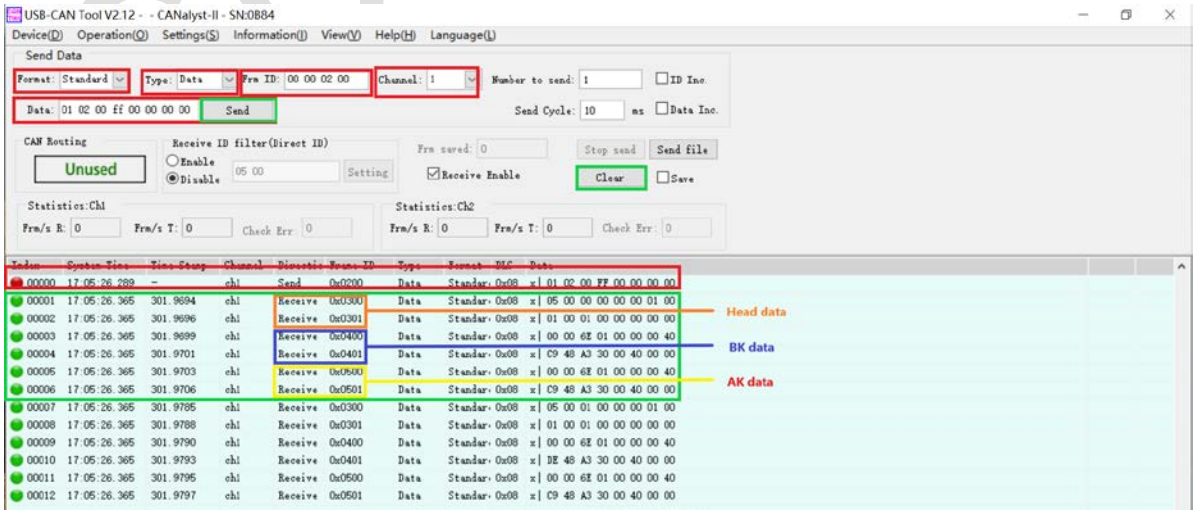

Figure 21.3 USB-CAN Tool

<5> Launch UART debugging tool, and connect to Calterah Radar Development Platform. After sending CAN message, logs will be printed through UART.

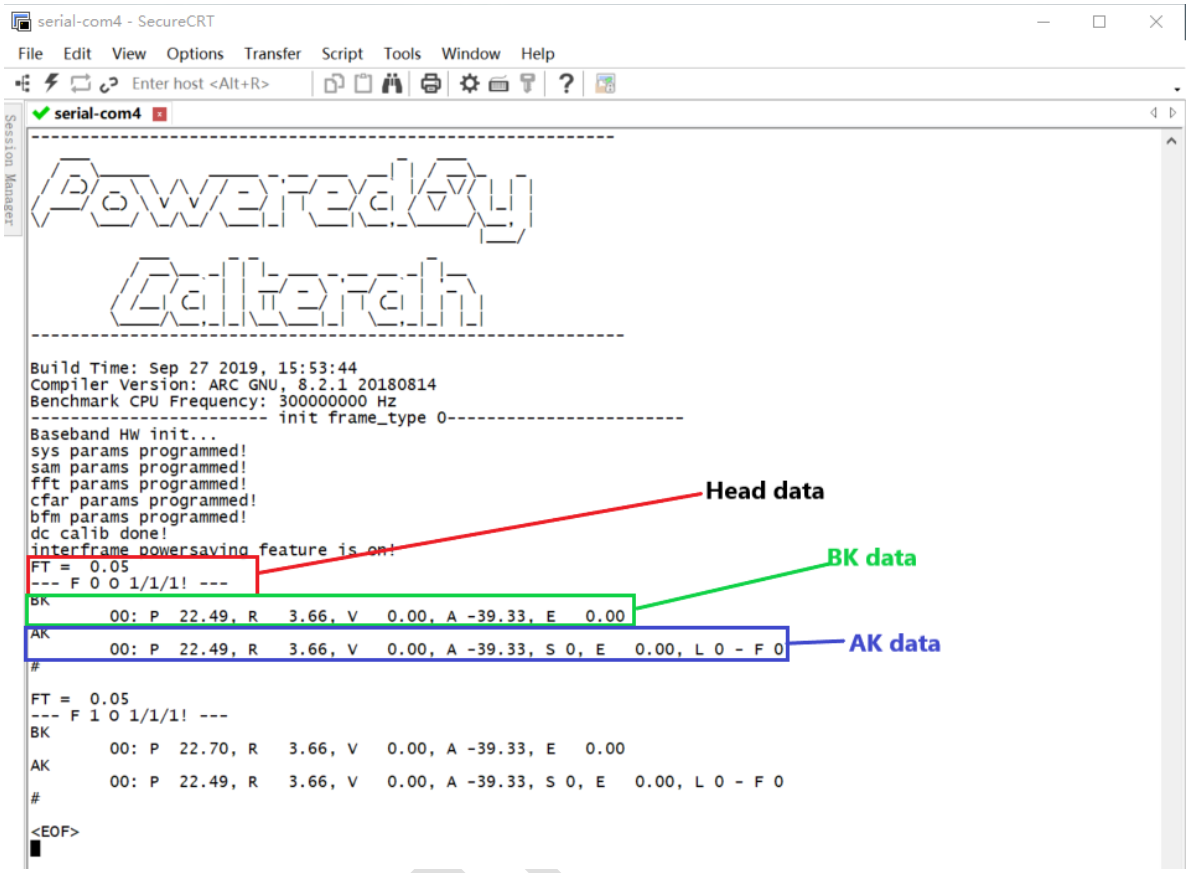

Figure 22 Uart Log

5.2 CAN FD Communication Verification

Hardware and software requirements:
<1> Calterah Radar Development Platform Alps
<2> Support CAN FD Analysis Tool: Kvaser USBcan Pro 2Xhs v2
<3> Personal computer
<4> Support CAN FD Analysis software: kvaser_canking_setup.exe

Notes:
Enabling the CAN FD requires setting the USE_CAN_FD to 1 in Alps firmware code.

Code Path:
embarc_osp\board\ref_design\ ref_design.mk
USE_CAN_FD ?= **1**

5.2.1 CAN FD Device Connection

Connect Micro-USB port on Alps with USB Type-A on a personal computer. The USB connection on Alps is used for power supply as well as sending logs through UART interface.

Connect CAN Analysis Tool to PC through a USB port.
Connect CAN 1 on CAN Analysis Tool (Kvaser USBcan Pro 2Xhs v2) with CAN 0 bus on Alps.

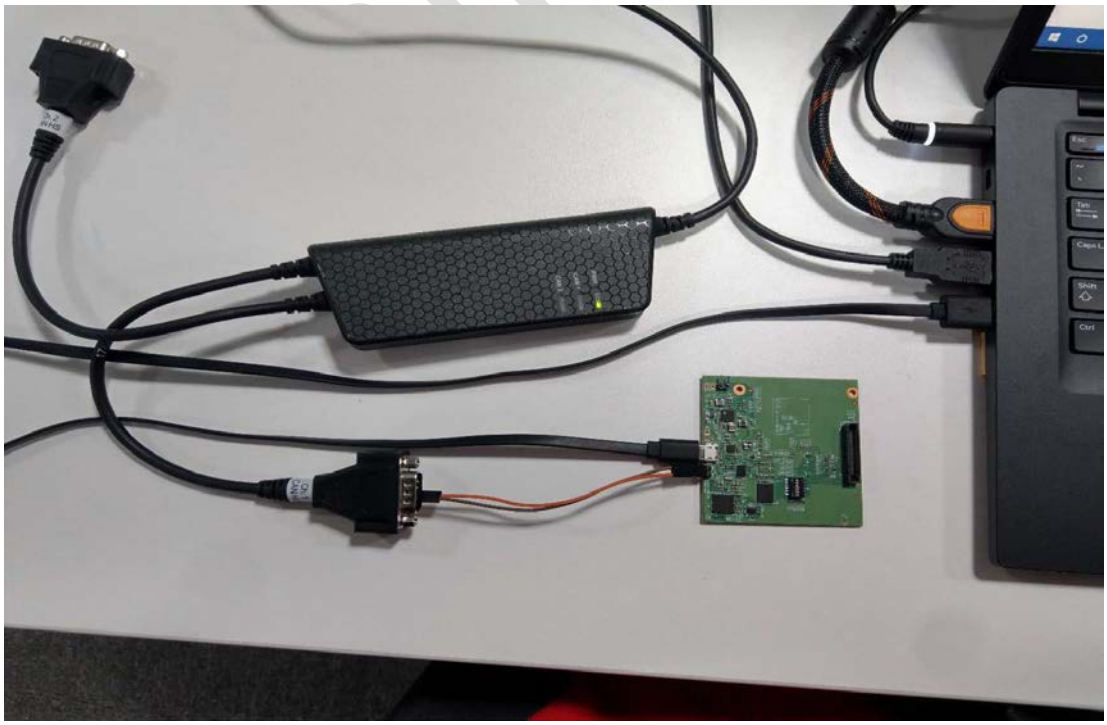

Figure 23 CAN FD Device Connection

5.1.2 CAN FD tool Software Installation

Install CAN analysis software kvaser_canking_setup.exe

5.1.3 Classic CAN Communication Debugging

<1> Launch Kvaser CanKing application. Select the CAN FD in “CAN Mode”.

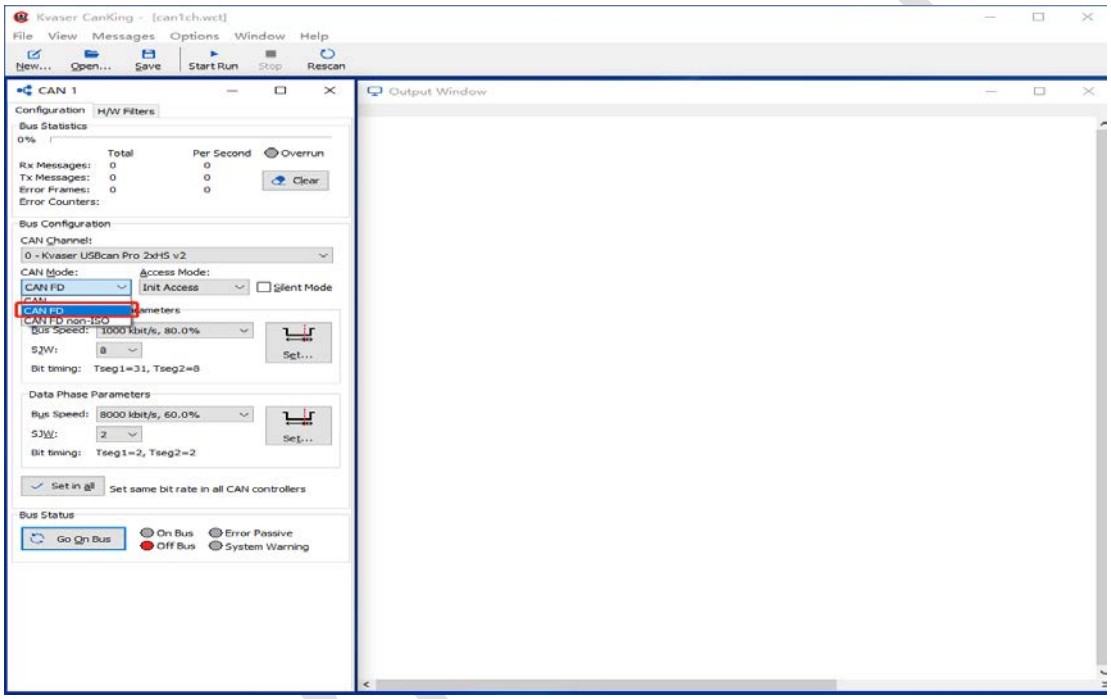

Figure 23.0 Kvaser CanKing

<2> Setting Phase Parameters (the same as the sampling point and the baud rate in firmware)

- Setting the Arbitration Phase Parameters

First, select a 500 kbit/s baud rate in the “Bus speed” list of the “Arbitration Phase Parameters”
Then click the “Set…” will pop a window.
Fine-tune the TSeg1 and the TSeg2 parameter so that the Sampling point of the arbitration phase becomes 70%

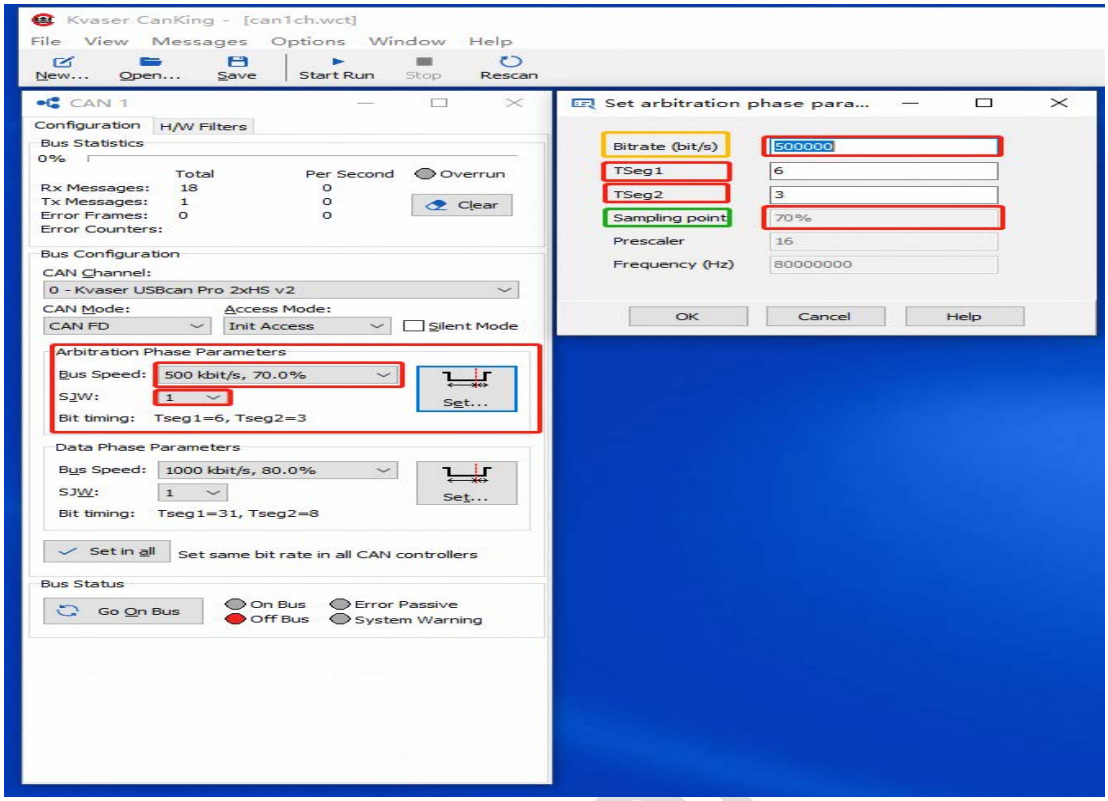

Figure 23.1 Kvaser CanKing

- Setting the Data Phase Parameters

First, select a 1000 kbit/s baud rate in the “Bus speed” list of the “Data Phase Parameters”
Then click the “Set···” will pop a window.
Fine-tune the TSeg1 and the TSeg2 parameter so that the Sampling point of the data phase becomes 80%

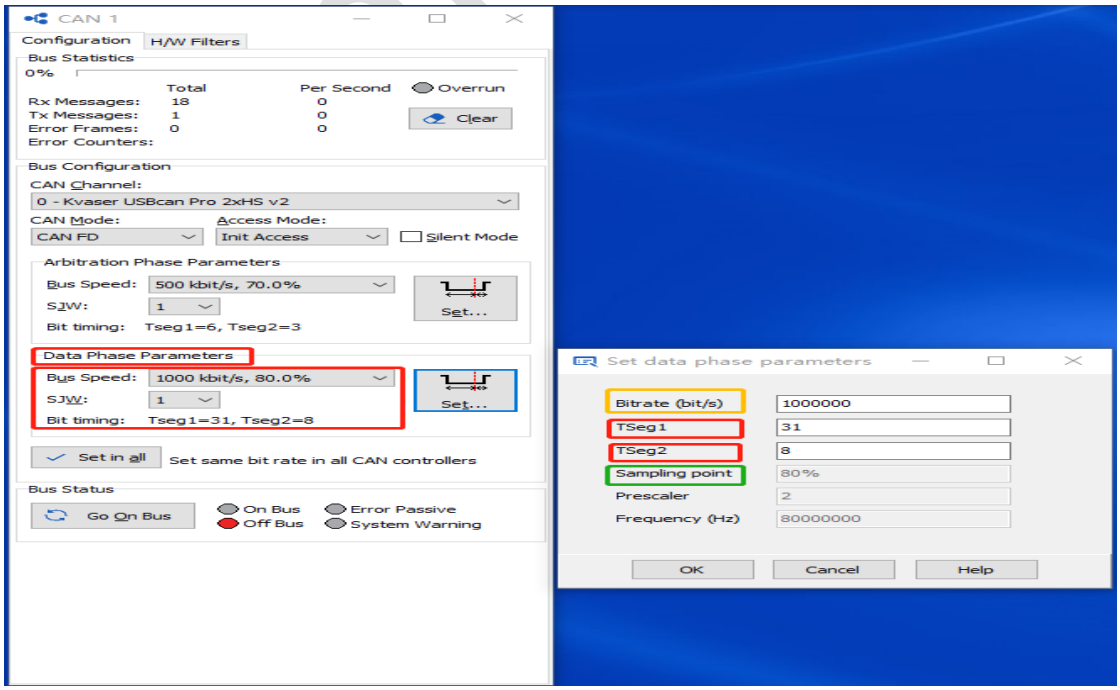

Figure 23.2 Kvaser CanKing

- Click “Go On Bus”

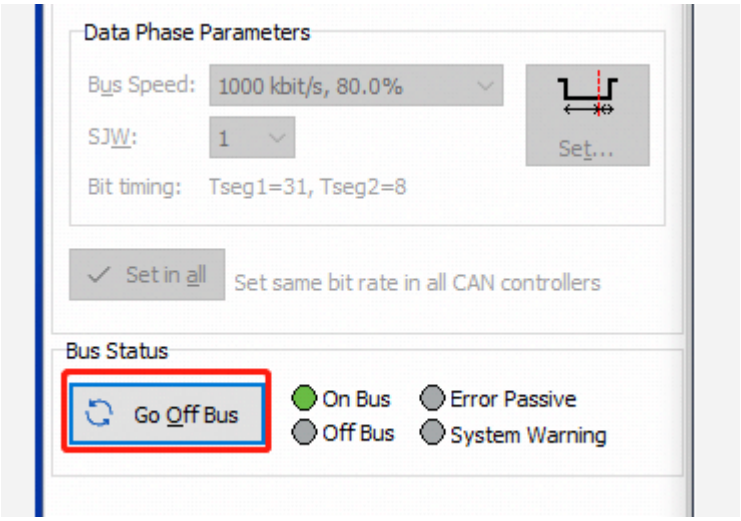

Figure 23.3 Kvaser CanKing

<3> CAN Message Configuration
Configure CAN message as follow.
CAN Identifier: 0x200

FDF: ☑

BRS: ☑

Channel: CAN 1
Data length: 16
Data (0…15): 010200ff 00000000 00000000 00000000
(The least 1 byte [01] represents start scanning. The second and third bytes [0002] represents the number of the frame for scanning is two. The highest one-byte [FF] represents outputting all tracking data including header, BK and AK.)

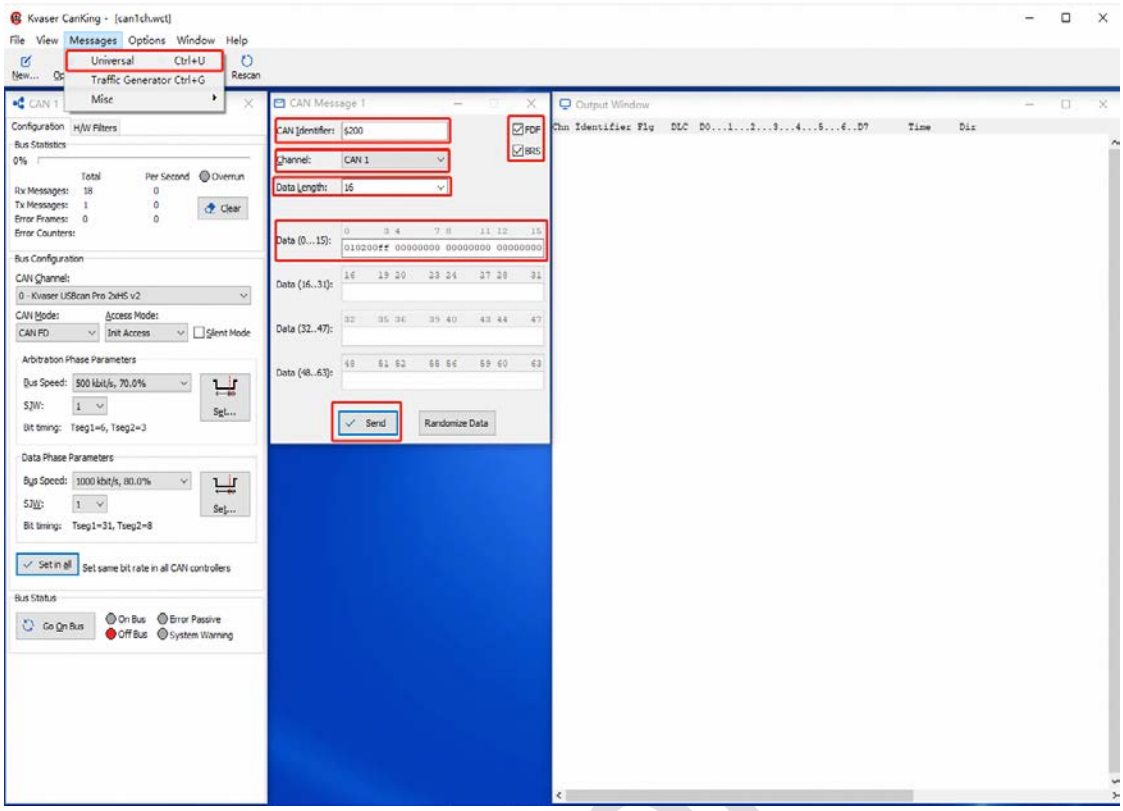

Figure 23.4 Kvaser CanKing

Press **Send** and the tracking data will be displayed in the output window.

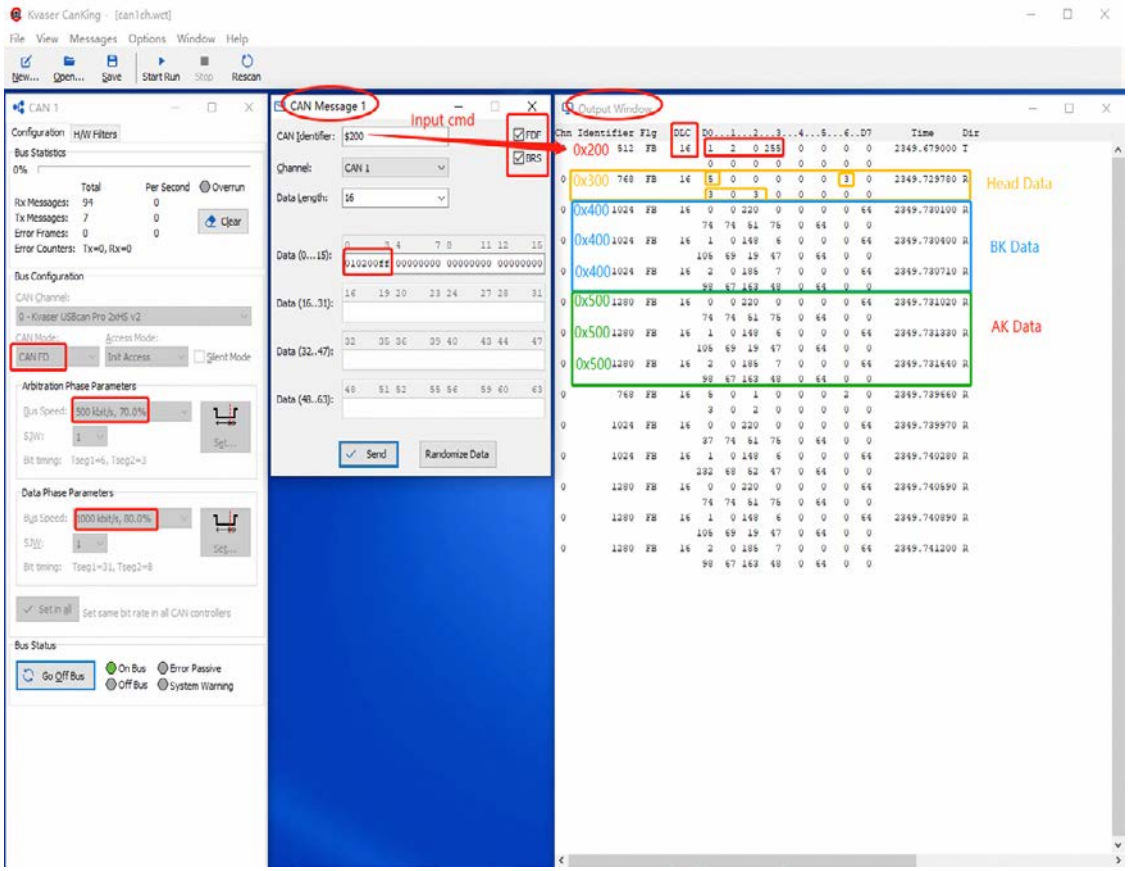

Figure 23.5 Kvaser CanKing