Allegro Hand CAN Protocol Specification

SIMLAB CO., LTD.
Version 1.0.0

Allegro Hand CAN Protocol Specification

Hyobong Bldg 2nd Fl, 1425-9 Seocho-Dong, Seocho-Gu,
Seoul 137-864, Korea

Copyright & Trademark Notice

Table of Contents

List of Tables

List of Code Samples

1. CAN Communication

1.1 Baud-Rate

The CAN communication baud-rate is 1Mbps.

1.2 Non-Periodic Communication

Messages can be sent to initialize or stop CAN communication.

1.3 Periodic Communication

The Allegro Hand control software attempts to communicate with the real or simulated hand at a regular control interval. Every 3 milliseconds the joint torques are calculated and the joint angles are updated.

2. CAN Frames

2.1 Standard CAN Packet

The standard CAN packet used for communication contains 8 bytes.

Code 1: CAN Packet Structure

```
typedef struct{
        unsigned char STD_EXT;
        unsigned long msg_id;           //message identifier
        unsigned char data_length;      //
        char          data[8];          // data array
} can_msg;
```

2.2. ID (Message Identifier)

The 4 byte integer CAN message is split into the command ID (26 bits), destination ID (3bits) and source ID (3 bits).

Table 1: CAN Message Identifiers

1 – 26	27 – 29	30 – 32
Command ID	DSTN. ID	Source ID

2.2.1 Command Identifiers

Table 2: CAN Message Command Identifiers

Variable name	Value	Description	Source	Destination
ID_CMD_SET_SYSTEM_ON	0x01	Start periodic communication	ID_DEVICE_MAIN	ID_COMMON
ID_CMD_SET_SYSTEM_OFF	0x02	Stop periodic communication	ID_DEVICE_MAIN	ID_COMMON
ID_CMD_SET_PERIOD	0x03	Set communication frequency	ID_DEVICE_MAIN	ID_COMMON
ID_CMD_SET_MODE_JOINT	0x04	Command Transmission Mode	ID_DEVICE_MAIN	ID_COMMON
ID_CMD_SET_MODE_TASK	0x05	Command Transmission Mode	ID_DEVICE_MAIN	ID_COMMON
ID_CMD_SET_TORQUE_1	0x06	Index finger (1) torque command	ID_DEVICE_MAIN	ID_COMMON
ID_CMD_SET_TORQUE_2	0x07	Middle finger (2) toque command	ID_DEVICE_MAIN	ID_COMMON
ID_CMD_SET_TORQUE_3	0x08	Pinky finger (3) torque command	ID_DEVICE_MAIN	ID_COMMON
ID_CMD_SET_TORQUE_4	0x09	Thumb torque command	ID_DEVICE_MAIN	ID_COMMON
ID_CMD_SET_POSITION_1	0x0a	*(unused)*		
ID_CMD_SET_POSITION_2	0x0b	*(unused)*		
ID_CMD_SET_POSITION_3	0x0c	*(unused)*		
ID_CMD_SET_POSITION_4	0x0d	*(unused)*		
ID_CMD_QUERY_STATE_DATA	0x0e	Request joint state	ID_DEVICE_MAIN	ID_COMMON
ID_CMD_QUERY_STATE_DATA	0x0e	Joint state response	ID_DEVICE_SUB_01 ID_DEVICE_SUB_02 ID_DEVICE_SUB_03 ID_DEVICE_SUB_04	ID_DEVICE_MAIN
ID_CMD_QUERY_CONTROL_DATA	0x0f	Joint state response	ID_DEVICE_SUB_01 ID_DEVICE_SUB_02 ID_DEVICE_SUB_03 ID_DEVICE_SUB_04	ID_DEVICE_MAIN

2.2.2 Source and Destination Identifiers

Table 3: Source and Destination Identifiers

Variable name	Value	Description
ID_COMMON	0x01	Allegro Hand
ID_DEVICE_MAIN	0x02	Control PC
ID_DEVICE_SUB_01	0x03	Index Finger
ID_DEVICE_SUB_02	0x04	Middle Finger
ID_DEVICE_SUB_03	0x05	Little Finger
ID_DEVICE_SUB_04	0x06	Thumb

3. Case-study: Softing CAN

In this chapter, sample code demonstrating the implementation of the CAN communication interface is provide. This is the foundation for Softing PCI CAN.

3.1 Opening the CAN Communication Channel

Code 2: Opening the CAN Communication Channel

```
char ch_name[256];
sprintf_s(ch_name, 256, "CAN-ACx-PCI_%d", ch);
INIL2_initialize_channel(&hCAN[ch-1], ch_name);

L2CONFIG L2Config;
L2Config.fBaudrate = 1000.0;
L2Config.bEnableAck = 0;
L2Config.bEnableErrorframe = 0;
L2Config.s32AccCodeStd = 0;
L2Config.s32AccMaskStd = 0;
L2Config.s32AccCodeXtd = 0;
L2Config.s32AccMaskXtd = 0;
L2Config.s32OutputCtrl = GET_FROM_SCIM;
L2Config.s32Prescaler = 1;
L2Config.s32Sam = 0;
L2Config.s32Sjw = 1;
L2Config.s32Tseg1 = 4;
L2Config.s32Tseg2 = 3;
L2Config.hEvent = (void*)-1;

CANL2_initialize_fifo_mode(hCAN[ch-1], &L2Config);
```


3.2 CAN Initialization

Code 3: Opening the CAN Communication Channel

```
long Txid;
unsigned char data[8];

Txid = ((unsigned long)ID_CMD_SET_PERIOD<<6) | ((unsigned long)ID_COMMON <<3) | ((unsigned long)ID_DEVICE_MAIN);
data[0] = (unsigned char)period_msec;
canWrite(hCAN, Txid, data, 1, STD);

Sleep(10);

Txid = ((unsigned long)ID_CMD_SET_MODE_TASK<<6) | ((unsigned long)ID_COMMON <<3) | ((unsigned long)ID_DEVICE_MAIN);
canWrite(hCAN, Txid, data, 0, STD);

Sleep(10);

Txid = ((unsigned long)ID_CMD_QUERY_STATE_DATA<<6) | ((unsigned long)ID_COMMON <<3) | ((unsigned long)ID_DEVICE_MAIN);
canWrite(hCAN, Txid, data, 0, STD);
```

3.3 Starting Periodic CAN Communication

When you start periodic CAN communication, joint angles are automatically updated according to the torque control input.

Code 4: Starting Periodic CAN Communication

```
long Txid;
unsigned char data[8];

Txid = ((unsigned long)ID_CMD_QUERY_STATE_DATA<<6) | ((unsigned long)ID_COMMON <<3) | ((unsigned long)ID_DEVICE_MAIN);
canWrite(hCAN[ch-1], Txid, data, 0, STD);

Sleep(10);

Txid = ((unsigned long)ID_CMD_SET_SYSTEM_ON<<6) | ((unsigned long)ID_COMMON <<3) | ((unsigned long)ID_DEVICE_MAIN);
canWrite(hCAN[ch-1], Txid, data, 0, STD);
```

3.4 Stopping Periodic CAN Communication

Code 5: Stopping Periodic CAN Communication

```
long Txid;
unsigned char data[8];

Txid = ((unsigned long)ID_CMD_SET_SYSTEM_OFF<<6) | ((unsigned long)ID_COMMON <<3) | ((unsigned long)ID_DEVICE_MAIN);
canWrite(hCAN[ch-1], Txid, data, 0, STD);
```


3.5 Transmitting Control Torques

Control inputs for the four joints in each finger should be packed in a single CAN frame. The sample code below demontrates how to encode four PWM inputs into an 8 byte data buffer and how to set the CAN frame ID properly.

Code 6: Transmitting Control Torques

```
long Txid;
unsigned char data[8];
float torque2pwm = 800.0f
short pwm[4] = {
                0.1*torque2pwm,
                0.1*torque2pwm,
                0.1*torque2pwm,
                0.1*torque2pwm
};

if (findex >= 0 && findex < 4)
{
                data[0] = (unsigned char)( (pwm[0] >> 8) & 0x00ff);
                data[1] = (unsigned char)(pwm[0] & 0x00ff);

                data[2] = (unsigned char)( (pwm[1] >> 8) & 0x00ff);
                data[3] = (unsigned char)(pwm[1] & 0x00ff);

                data[4] = (unsigned char)( (pwm[2] >> 8) & 0x00ff);
                data[5] = (unsigned char)(pwm[2] & 0x00ff);

                data[6] = (unsigned char)( (pwm[3] >> 8) & 0x00ff);
                data[7] = (unsigned char)(pwm[3] & 0x00ff);

                Txid = ((unsigned long)(ID_CMD_SET_TORQUE_1 + findex)<<6) | ((unsigned long)ID_COMMON <<3) | ((unsigned long)ID_DEVICE_MAIN);
                canWrite(hCAN, Txid, data, 8, STD);
}
```

3.6 Receiving Joint Angles

Each finger consists of four joints. The joint angles for those four joints can be received via one CAN packet. The sample code below demonstrates the method for decoding the data buffer and reading the joint angles.

The sample code assumes that when fingers are in their zero positions, the joint angles from the can packet are *32768*. In practice, users should perform experiments and introduce offsets to obtain the zero position.

Code 7: Receiving Joint Angles

```
char cmd;
char src;
char des;
int len;
unsigned char data[8];
int ret;
can_msg msg;
PARAM_STRUCT param;

ret = CANL2_read_ac(hCAN, &param);

switch (ret)
{
case CANL2_RA_DATAFRAME:
		msg.msg_id = param.Ident;
		msg.STD_EXT = STD;
		msg.data_length = param.DataLength;

		msg.data[0] = param.RCV_data[0];
		msg.data[1] = param.RCV_data[1];
		msg.data[2] = param.RCV_data[2];
		msg.data[3] = param.RCV_data[3];
		msg.data[4] = param.RCV_data[4];
		msg.data[5] = param.RCV_data[5];
		msg.data[6] = param.RCV_data[6];
		msg.data[7] = param.RCV_data[7];

		break;
}

cmd = (char)( (msg.msg_id >> 6) & 0x1f );
des = (char)( (msg.msg_id >> 3) & 0x07 );
src = (char)( msg.msg_id & 0x07 );
len = (int)( msg.data_length );
for(int nd=0; nd<len; nd++)
		data[nd] = msg.data[nd];

switch (cmd)
{
case ID_CMD_QUERY_CONTROL_DATA:
		{
				if (id_src >= ID_DEVICE_SUB_01 && id_src <= ID_DEVICE_SUB_04)
				{
						int temp_pos[4]; // raw angle data
						float ang[4]; // degree
						float q[4]; // radian

						temp_pos[0] = (int)(data[0] | (data[1] << 8));
						temp_pos[1] = (int)(data[2] | (data[3] << 8));
						temp_pos[2] = (int)(data[4] | (data[5] << 8));
						temp_pos[3] = (int)(data[6] | (data[7] << 8));

						ang[0] = ((float)(temp_pos[0]-32768)*(333.3f/65536.0f))*(1);
						ang[1] = ((float)(temp_pos[1]-32768)*(333.3f/65536.0f))*(1);
						ang[2] = ((float)(temp_pos[2]-32768)*(333.3f/65536.0f))*(1);
						ang[3] = ((float)(temp_pos[3]-32768)*(333.3f/65536.0f))*(1);

						q[0] = (3.141592f/180.0f) * ang[0];
						q[1] = (3.141592f/180.0f) * ang[1];
						q[2] = (3.141592f/180.0f) * ang[2];
						q[3] = (3.141592f/180.0f) * ang[3];
				}
		}
}
```

