**Allegro Hand CAN Protocol Specification**

SIMLAB CO., LTD.

Version 1.0.0

**Allegro Hand CAN Protocol Specification**

Copyright © 2008-2012 SimLab Co., Ltd.

Hyobong Bldg 2nd Fl, 1425-9 Seocho-Dong, Seocho-Gu,

Seoul 137-864, Korea

Copyright & Trademark Notice

Allegro, the Allegro logo and all related files and documentation

are Copyright © 2008-2012 SimLab Co., Ltd. All rights reserved.

Allegro is a trademark of SimLab Co., Ltd. All other trademarks or registered trademarks mentioned.

Table of Contents

[1. CAN Communication 2](#_Toc335154743)

[1.1 Baud-Rate 2](#_Toc335154744)

[1.2 Non-Periodic Communication 2](#_Toc335154745)

[1.3 Periodic Communication 2](#_Toc335154746)

[2. CAN Frames 3](#_Toc335154747)

[2.1 Standard CAN Packet 3](#_Toc335154748)

[2.2. ID (Message Identifier) 4](#_Toc335154749)

[2.2.1 Command Identifiers 4](#_Toc335154750)

[2.2.2 Source and Destination Identifiers 4](#_Toc335154751)

[3. Case-study: Softing CAN 4](#_Toc335154752)

[3.1 Opening the CAN Communication Channel 5](#_Toc335154753)

[3.2 CAN Initialization 6](#_Toc335154754)

[3.3 Starting Periodic CAN Communication 6](#_Toc335154755)

[3.4 Stopping Periodic CAN Communication 6](#_Toc335154756)

[3.5 Transmitting Control Torques 7](#_Toc335154757)

[3.6 Receiving Joint Angles 8](#_Toc335154758)

List of Tables

Table 1: CAN Message Identifiers 4

Table 2: CAN Message Command Identifiers 4

Table 3: Source and Destination Identifiers 4

List of Code Samples

Code 1: CAN Packet Structure 3

Code 2: Opening the CAN Communication Channel 5

Code 3: Opening the CAN Communication Channel 6

Code 4: Starting Periodic CAN Communication 6

Code 5: Stopping Periodic CAN Communication 6

Code 6: Transmitting Control Torques 7

# 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 | 8 | 16 | 24 | 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** | **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 |

# 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 : 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 : 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 : 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 : 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 : 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 : 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];  }  }  } |