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
   1.1 Baud-Rate .......................................................................................................................................................... 2
   1.2 Non-Periodic Communication ...................................................................................................................... 2
   1.3 Periodic Communication .............................................................................................................................. 2
2. CAN Frames ............................................................................................................................................................. 3
   2.1 Standard CAN Packet ...................................................................................................................................... 3
   2.2 ID (Message Identifier) .................................................................................................................................. 4
      2.2.1 Command Identifiers ........................................................................................................................ 4
      2.2.2 Source and Destination Identifiers ..................................................................................................... 4
3. Case-study: Softing CAN ......................................................................................................................................... 4
   3.1 Opening the CAN Communication Channel ................................................................................................. 5
   3.2 CAN Initialization ........................................................................................................................................... 6
   3.3 Starting Periodic CAN Communication ........................................................................................................ 6
   3.4 Stopping Periodic CAN Communication ....................................................................................................... 6
   3.5 Transmitting Control Torques ...................................................................................................................... 7
   3.6 Receiving Joint Angles .................................................................................................................................. 8
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**

```c
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

<table>
<thead>
<tr>
<th>Command ID</th>
<th>DSTN. ID</th>
<th>Source ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>8</td>
<td>16</td>
</tr>
</tbody>
</table>

2.2.1 Command Identifiers

Table 2: CAN Message Command Identifiers

<table>
<thead>
<tr>
<th>Variable name</th>
<th>Value</th>
<th>Description</th>
<th>Source</th>
<th>Destination</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID_CMD_SET_SYSTEM_ON</td>
<td>0x01</td>
<td>Start periodic communication</td>
<td>ID_DEVICE_MAIN</td>
<td>ID_COMMON</td>
</tr>
<tr>
<td>ID_CMD_SET_SYSTEM_OFF</td>
<td>0x02</td>
<td>Stop periodic communication</td>
<td>ID_DEVICE_MAIN</td>
<td>ID_COMMON</td>
</tr>
<tr>
<td>ID_CMD_SET_PERIOD</td>
<td>0x03</td>
<td>Set communication frequency</td>
<td>ID_DEVICE_MAIN</td>
<td>ID_COMMON</td>
</tr>
<tr>
<td>ID_CMD_SET_MODE_JOINT</td>
<td>0x04</td>
<td>Command Transmission Mode</td>
<td>ID_DEVICE_MAIN</td>
<td>ID_COMMON</td>
</tr>
<tr>
<td>ID_CMD_SET_MODE_TASK</td>
<td>0x05</td>
<td>Command Transmission Mode</td>
<td>ID_DEVICE_MAIN</td>
<td>ID_COMMON</td>
</tr>
<tr>
<td>ID_CMD_SET_TORQUE_1</td>
<td>0x06</td>
<td>Index finger (1) torque command</td>
<td>ID_DEVICE_MAIN</td>
<td>ID_COMMON</td>
</tr>
<tr>
<td>ID_CMD_SET_TORQUE_2</td>
<td>0x07</td>
<td>Middle finger (2) torque command</td>
<td>ID_DEVICE_MAIN</td>
<td>ID_COMMON</td>
</tr>
<tr>
<td>ID_CMD_SET_TORQUE_3</td>
<td>0x08</td>
<td>Pinky finger (3) torque command</td>
<td>ID_DEVICE_MAIN</td>
<td>ID_COMMON</td>
</tr>
<tr>
<td>ID_CMD_SET_TORQUE_4</td>
<td>0x09</td>
<td>Thumb torque command</td>
<td>ID_DEVICE_MAIN</td>
<td>ID_COMMON</td>
</tr>
<tr>
<td>ID_CMD_SET_POSITION_1</td>
<td>0xa</td>
<td>(unused)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ID_CMD_SET_POSITION_2</td>
<td>0xb</td>
<td>(unused)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ID_CMD_SET_POSITION_3</td>
<td>0xc</td>
<td>(unused)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ID_CMD_SET_POSITION_4</td>
<td>0xd</td>
<td>(unused)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ID_CMD_QUERY_STATE_DATA</td>
<td>0xe</td>
<td>Request joint state</td>
<td>ID_DEVICE_SUB_01</td>
<td>ID_COMMON</td>
</tr>
<tr>
<td>ID_CMD_QUERY_STATE_DATA</td>
<td>0xe</td>
<td>Joint state response</td>
<td>ID_DEVICE_SUB_02</td>
<td>ID_DEVICE_MAIN</td>
</tr>
<tr>
<td>ID_CMD_QUERY_STATE_DATA</td>
<td>0xe</td>
<td>Joint state response</td>
<td>ID_DEVICE_SUB_03</td>
<td>ID_DEVICE_SUB_04</td>
</tr>
<tr>
<td>ID_CMD_QUERY_STATE_DATA</td>
<td>0xe</td>
<td>Joint state response</td>
<td>ID_DEVICE_SUB_01</td>
<td>ID_DEVICE_SUB_02</td>
</tr>
<tr>
<td>ID_CMD_QUERY_STATE_DATA</td>
<td>0xe</td>
<td>Joint state response</td>
<td>ID_DEVICE_SUB_03</td>
<td>ID_DEVICE_SUB_04</td>
</tr>
<tr>
<td>ID_CMD_QUERY_CONTROL_DATA</td>
<td>0xf</td>
<td>Joint state response</td>
<td>ID_DEVICE_SUB_01</td>
<td>ID_DEVICE_SUB_02</td>
</tr>
<tr>
<td>ID_CMD_QUERY_CONTROL_DATA</td>
<td>0xf</td>
<td>Joint state response</td>
<td>ID_DEVICE_SUB_03</td>
<td>ID_DEVICE_SUB_04</td>
</tr>
<tr>
<td>ID_CMD_QUERY_CONTROL_DATA</td>
<td>0xf</td>
<td>Joint state response</td>
<td>ID_DEVICE_SUB_01</td>
<td>ID_DEVICE_SUB_02</td>
</tr>
<tr>
<td>ID_CMD_QUERY_CONTROL_DATA</td>
<td>0xf</td>
<td>Joint state response</td>
<td>ID_DEVICE_SUB_03</td>
<td>ID_DEVICE_SUB_04</td>
</tr>
</tbody>
</table>

2.2.2 Source and Destination Identifiers

Table 3: Source and Destination Identifiers

<table>
<thead>
<tr>
<th>Variable name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID_COMMON</td>
<td>0x01</td>
<td>Allegro Hand</td>
</tr>
<tr>
<td>ID_DEVICE_MAIN</td>
<td>0x02</td>
<td>Control PC</td>
</tr>
<tr>
<td>ID_DEVICE_SUB_01</td>
<td>0x03</td>
<td>Index Finger</td>
</tr>
<tr>
<td>ID_DEVICE_SUB_02</td>
<td>0x04</td>
<td>Middle Finger</td>
</tr>
<tr>
<td>ID_DEVICE_SUB_03</td>
<td>0x05</td>
<td>Little Finger</td>
</tr>
<tr>
<td>ID_DEVICE_SUB_04</td>
<td>0x06</td>
<td>Thumb</td>
</tr>
</tbody>
</table>

3. Case-study: Softing CAN
In this chapter, sample code demonstrating the implementation of the CAN communication interface is provided. This is the foundation for Softing PCI CAN.

3.1 Opening the CAN Communication Channel

Code 2: Opening the CAN Communication Channel

```c
char ch_name[256];
sprintf(ch_name, 256, "CAN-ACx-PCI_%d", ch);
INIL2_initialize_channel(hICAN[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_FR_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(hICAN[ch-1], &L2Config);
```
3.2 CAN Initialization

**Code 3:** Opening the CAN Communication Channel

```c
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

```c
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

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

Txid = ((unsigned long)ID_CMD_SET_SYSTEM_OFF<<6) | ((unsigned long)ID_COMMON <<3) | ((unsigned long)IDDEVICE 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 demonstrates 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**

```c
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**

```c
char cmd;
char src;
char des;
int len;
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 );
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));

            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];
        }
    }
    
    }
```