Motion Mind DC Motor Controller 1.0 Reference Manual

Generated by Doxygen 1.4.6

Tue Apr 11 14:51:00 2006

Contents

Chapter 1

Motion Mind DC Motor Controller 1.0 Hierarchical Index

1.1 Motion Mind DC Motor Controller 1.0 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 2

Motion Mind DC Motor Controller 1.0 Class Index

2.1 Motion Mind DC Motor Controller 1.0 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

Motion Mind DC Motor Controller 1.0 File Index

3.1 Motion Mind DC Motor Controller 1.0 File List

Here is a list of all files with brief descriptions:

Chapter 4

Motion Mind DC Motor Controller 1.0 Class Documentation

4.1 MMClient Class Reference

A non-blocking client to MMServer.

`#include <MMClientServer.hpp>`

Public Member Functions

- MMClient (char *shmName="mmc0")
- ~MMClient ()
- int isServerStatusOk ()
- void changeSpeed (int16_t speed)
- void moveToAbsolute (int32_t position)
- void moveToRelative (int32_t position)
- void moveAtVelocity (int16_t velocity)
- int32_t getPosition ()

4.1.1 Detailed Description

A non-blocking client to MMServer.

Use an object of this class to command the Motion Mind Controller through a server. The functionality is restricted to setting speed and position using modes 3 or 4 of the controller (serial open loop, and serial closed loop PID, respectively)

Example Program:

```
//============================================================================
// Package: Motion Mind DC Motor Controller
// Authors: Vilas Kumar Chitrakaran, Nitendra Nath
// Start Date: Sun Jan 15 15:36:46 EST 2006
// ----------------------------------------------------------------------
// File: MMClientServer.t.cpp
//============================================================================
```

```
#include <stdio.h>
#include "MMClientServer.hpp"
#include <pthread.h>
#include <unistd.h>
#include <math.h>


//========================================================================
// client
//========================================================================
int client()
{
 int t = 0;
 MMClient mmClient("/mmc0");
 if(mmClient.isServerStatusOk() != 0) {
  fprintf(stderr, "[client]: ERROR in server.\n");
  return -1;
 }

 while(1) {
  if(mmClient.isServerStatusOk() != 0) {
   fprintf(stderr, "[client]: ERROR in server.\n");
   return -1;
  }
  mmClient.moveAtVelocity((int)(sin((float)t) * 100.0));
  fprintf(stdout, "%d\n", mmClient.getPosition());
  t+=1;
  sleep(1);
 }
 return 0;
}

//========================================================================
// server
//========================================================================
void *server(void *arg)
{
 MMServer mmServer("/mmc0", "/dev/ser1", 19200, 1);
 if(mmServer.getStatusCode() != 0) {
  fprintf(stderr, "[server]: %s\n", mmServer.getStatusMessage());
  return (void *)(-1);
 }
 mmServer.doMainLoop(10);
 return 0;
}


//============================================================================
// main function
//============================================================================
int main()
{
 pthread_t threadId;
 pthread_create(&threadId, NULL, &server, NULL);
 sleep(1);
 client();
 return 0;
}
```

4.1.2 Constructor & Destructor Documentation

4.1.2.1 MMClient::MMClient (char * *shmName* = "mmc0")

The default constructor. Initiates connection with server

Parameters:

shmName The server (MMServer class) identifier

4.1.2.2 MMClient::~MMClient ()

The default destructor. Disconnects from server.

4.1.3 Member Function Documentation

4.1.3.1 void MMClient::changeSpeed (int16_t *speed*)

Modifies speed and direction of motor when controller is operated in mode 3 (serial open loop control).

Parameters:

speed Value in the range -1023 to +1023. The direction is determined by the sign (- reverse, + forward) and the the absolute value sets the duty cycle of the input voltage (for example -512 => motor is driven in reverse with 50% duty cycle). The timer register (MMReg_timer) and the velocity limit register (MMReg_velocityLimit) determines the ramp up time required to achieve the new speed.

4.1.3.2 int32_t MMClient::getPosition ()

The client continously receives an update for actual motor position. This position data is updated only as fast as the server can provide this information, and hence, not guaranteed to be exactly correct. Use this method to get a rough estimate for motor position.

4.1.3.3 int MMClient::isServerStatusOk ()

Returns:

0 if server status is OK, else -1.

4.1.3.4 void MMClient::moveAtVelocity (int16_t *velocity*)

Specify velocity when operating in mode 4 (serial closed loop). The PID filter must be tuned to get best results.

Parameters:

velocity The desired velocity as a 2's compliment integer. The velocity measurement in the controller occurs over a 5 ms period. Hence, the desired velocity setting can be computed from the following equation: velocity = (MotorShaftRotations/Second * EncoderCountPerRotation * GearRatio)/50. Use negative values for motion in reverse direction.

4.1.3.5 void MMClient::moveToAbsolute (int32_t *position*)

Specify an absolute motor position when operating in mode 4 (serial closed-loop). Ensure that the PID filter is tuned, and note that the encoder is decoded on a 4:1 ratio (i.e., if the encoder gives 500 pulses for 1 rotation of the shaft, send 2000 to rotate motor shaft by one full rotation.

Parameters:

position Absolute position as a 32 bit 2's compliment number. Set velocity limit bit in the function register (MMFunc_velocityLimit) and load velocity limit register (mmcreg_-velocityLimit) to limit average velocity during motion.

4.1.3.6 void MMClient::moveToRelative (int32_t *position*)

Specify movement to a position relative to current position, when operating in mode 4 (serial closed loop). Ensure that the PID filter is tuned, and note that the encoder is decoded on a 4:1 ratio (i.e., if the encoder gives 500 pulses for 1 rotation of the shaft, send 2000 to rotate motor shaft by one full rotation.

Parameters:

position Relative position as a 32 bit 2's compliment number. Set velocity limit bit in the function register (MMFunc_velocityLimit) and load velocity limit register (MMReg_-velocityLimit) to limit average velocity during motion.

The documentation for this class was generated from the following file:

- MMClientServer.hpp

4.2 MMCommand Struct Reference

`#include <MMClientServer.hpp>`

Public Attributes

- sem_t guard
- char command
- int32_t data
- int32_t position
- char status

4.2.1 Member Data Documentation

4.2.1.1 char MMCommand::command

4.2.1.2 int32_t MMCommand::data

4.2.1.3 sem_t MMCommand::guard

4.2.1.4 int32_t MMCommand::position

4.2.1.5 char MMCommand::status

The documentation for this struct was generated from the following file:

- MMClientServer.hpp

4.3 MMData Struct Reference

`#include <MotionMind.hpp>`

Public Attributes

- int32_t position
- int16_t velocityLimit
- int8_t velocityFf
- int16_t function
- int16_t pTerm
- int16_t iTerm
- int16_t dTerm
- int8_t address
- int8_t pidScalar
- int8_t timer
- int16_t rcMax
- int16_t rcMin
- int16_t rcBand
- int16_t rcCount
- int16_t velocity
- int32_t time
- int16_t status
- int8_t revision
- int8_t mode
- int16_t analogCon
- int16_t analogFbck
- int16_t pwmOut
- int32_t indexPos

4.3.1 Member Data Documentation

4.3.1.1 int8_t MMData::address

4.3.1.2 int16_t MMData::analogCon

4.3.1.3 int16_t MMData::analogFbck

4.3.1.4 int16_t MMData::dTerm

4.3.1.5 int16_t MMData::function

4.3.1.6 int32_t MMData::indexPos

4.3.1.7 int16_t MMData::iTerm

4.3.1.8 int8_t MMData::mode

4.3.1.9 int8_t MMData::pidScalar

4.3.1.10 int32_t MMData::position

4.3.1.11 int16_t MMData::pTerm

4.3.1.12 int16_t MMData::pwmOut

4.3.1.13 int16_t MMData::rcBand

4.3.1.14 int16_t MMData::rcCount

4.3.1.15 int16_t MMData::rcMax

4.3.1.16 int16_t MMData::rcMin

4.3.1.17 int8_t MMData::revision

4.3.1.18 int16_t MMData::status

4.3.1.19 int32_t MMData::time

4.3.1.20 int8_t MMData::timer

4.3.1.21 int16_t MMData::velocity

4.3.1.22 int8_t MMData::velocityFf

4.3.1.23 int16_t MMData::velocityLimit

The documentation for this struct was generated from the following file:

- MotionMind.hpp

4.4 MMServer Class Reference

Provides a server for motion mind controller.

`#include <MMClientServer.hpp>`

Public Member Functions

- MMServer (const char *serverName, const char *port, int baud, int address, bool verbose=true)
- ~MMServer ()
- int getStatusCode () const
- const char * getStatusMessage () const
- int setPIDGains (int p, int i, int d)
- void doMainLoop (int updateDelay)

4.4.1 Detailed Description

Provides a server for motion mind controller.

A client (MMClient class) can connect to an object of this class to set desired velocities and positions through a shared memory interface. Use a separate program to set internal registers in the board.

Example Program: See example program for MMClient

4.4.2 Constructor & Destructor Documentation

4.4.2.1 MMServer::MMServer (const char * *serverName*, const char * *port*, int *baud*, int *address*, bool *verbose* = `true`)

Default constructor

Parameters:

serverName Name of the server

port The serial port name (example /dev/ttyS0, /dev/ser1)

baud The baud select and motor direction lines are common in connector J4. When the desired operating mode is 2 (button mode), the baud rate is selected based on motor direction. Otherwise, it is advised to set baud rate to 19.2 KBPS. Specify either 9600 or 19200 for the selected baud rate here.

address Multiple contollers may be daisy chained. Specify which device is being controlled by an instance of this object using this parameter. (address=1 if only one device present).

verbose Print device info if set to true.

4.4.2.2 MMServer::~MMServer ()

Destructor.

4.4.3 Member Function Documentation

4.4.3.1 void MMServer::doMainLoop (int *updateDelay*)

Main loop of the server.

Parameters:
updateDelay Delay (ms) between every update cycle.

4.4.3.2 int MMServer::getStatusCode () const

Returns:
0 if server status is OK, else -1.

4.4.3.3 const char* MMServer::getStatusMessage () const

Returns:
status message.

4.4.3.4 int MMServer::setPIDGains (int *p*, int *i*, int *d*)

Set the P, I and D gains

Returns:
0 on success, -1 on error.

The documentation for this class was generated from the following file:

- MMClientServer.hpp

4.5 MotionMind Class Reference

A driver for the Solutions Cubed Motion Mind DC motor controller Revision 4.

`#include <MotionMind.hpp>`

Public Member Functions

- MotionMind ()
- ~MotionMind ()
- int init (const char *port, int baud, int address, bool verbose=true)
- int changeSpeed (int16_t speed)
- int moveToAbsolute (int32_t position)
- int moveToRelative (int32_t position)
- int moveAtVelocity (int16_t velocity)
- int writeRegister (MMReg_t reg, int32_t val)
- int writeStoreRegister (MMReg_t reg, int32_t val)
- int32_t readRegister (MMReg_t reg)
- int restore ()
- int reset ()

4.5.1 Detailed Description

A driver for the Solutions Cubed Motion Mind DC motor controller Revision 4.

The documentation in this file supplements the technical manual for the motion mind controller which can be obtained from http://www.solutions-cubed.com/solutions%20cubed/MM1_-2005.htm . This driver is designed for the controller released with revision 4 of the above technical manual (late 2005). This driver class provides a RS232 communication interface to the controller, and is most useful for modes 3 (serial open-loop control) and 4 (serial PID position control) specified in the controller manual. The user must select the mode of operation by setting jumpers J5, and RS232 baud rate through J2.

Example Program:

```
//=====================================================================
// Package: Motion Mind DC Motor Controller
// Authors: Vilas Kumar Chitrakaran, Nitendra Nath
// Start Date: Sun Jan 15 15:36:46 EST 2006
// ---------------------------------------------------------------------
// File: MotionMind.t.cpp
// Example program for the class MotionMind.
//=====================================================================


#include <stdio.h>
#include <unistd.h>
#include "MotionMind.hpp"

//=====================================================================
// main - An example PID control program
//=====================================================================
int main()
{
 MotionMind controller;
 int ret = 0;
```

```
 // connect to the controller
 ret = controller.init("/dev/ser1", 19200, 1);
 if(ret == -1) {
  fprintf(stderr, "Connection error\n");
  return(-1);
 }

 // set gains p = 6000, I = 35, D = 200
 if( controller.writeRegister(MMReg_pTerm, 6000) == -1 ) {
  fprintf(stderr, "[main] ERROR setting P gain\n");
 }
 if( controller.writeRegister(MMReg_iTerm, 35) == -1 ) {
  fprintf(stderr, "[main] ERROR setting I gain\n");
 }
 if( controller.writeRegister(MMReg_dTerm, 200) == -1 ) {
  fprintf(stderr, "[main] ERROR setting D gain\n");
 }

 // read current position
 fprintf(stdout, "current position: %d\n", (unsigned int) controller.readRegister(MMReg_position));

 // send desired position (in encoder counts x 4)
 if( controller.moveToAbsolute(5000) == -1 ) {
  fprintf(stderr, "[main] ERROR sending desired position\n");
 }

 // wait
 sleep(4);

 // bye (controller automatically resets and stops motor on exit)

 return 0;
}
```

4.5.2 Constructor & Destructor Documentation

4.5.2.1 MotionMind::MotionMind ()

The default constructor does nothing. Call init() to connect to the device.

4.5.2.2 MotionMind::~MotionMind ()

The default destructor. Frees resources.

4.5.3 Member Function Documentation

4.5.3.1 int MotionMind::changeSpeed (int16_t *speed*)

Modifies speed and direction of motor when controller is operated in mode 3 (serial open loop control). The device takes between 5 - 10 ms to respond to this command.

Parameters:

speed Value in the range -1023 to +1023. The direction is determined by the sign (- reverse, + forward) and the the absolute value sets the duty cycle of the input voltage (for example -512 => motor is driven in reverse with 50% duty cycle). The timer register (MMReg_timer) and the velocity limit register (MMReg_velocityLimit) determines the ramp up time required to achieve the new speed.

Returns:
0 on success, -1 on error.

4.5.3.2 int MotionMind::init (const char * *port*, int *baud*, int *address*, bool *verbose* = `true`)

Connects to the device through a specified serial port. Only RS232 binary data communication is supported. NOTE: This method must be called before calling any other method to interact with the hardware.

- Make sure jumper pins in J2 are set to 19.2 KBPS baud rate and Binary mode.
- Make sure that the PC's serial port is hooked to RS232 lines of the device.
- Ensure that mode of operation is selected through J5.

Parameters:
port The serial port name (example /dev/ttyS0, /dev/ser1)

baud The baud select and motor direction lines are common in connector J4. When the desired operating mode is 2 (button mode), the baud rate is selected based on motor direction. Otherwise, it is advised to set baud rate to 19.2 KBPS. Specify either 9600 or 19200 for the selected baud rate here.

address Multiple contollers may be daisy chained. Specify which device is being controlled by an instance of this object using this parameter. (address=1 if only one device present).

verbose Print device info if set to true.

Returns:
0 on success, -1 in case of error.

4.5.3.3 int MotionMind::moveAtVelocity (int16_t *velocity*)

Specify velocity when operating in mode 4 (serial closed loop). The PID filter must be tuned to get best results. The device takes between 5 - 10 ms to respond to this command.

Parameters:
velocity The desired velocity as a 2's compliment integer. The velocity measurement in the controller occurs over a 5 ms period. Hence, the desired velocity setting can be computed from the following equation: velocity = (MotorShaftRotations/Second * EncoderCountPerRotation * GearRatio)/50. Use negative values for motion in reverse direction.

Returns:
0 on success, -1 on error.

4.5.3.4 int MotionMind::moveToAbsolute (int32_t *position*)

Specify an absolute motor position when operating in mode 4 (serial closed-loop). Ensure that the PID filter is tuned, and note that the encoder is decoded on a 4:1 ratio (i.e., if the encoder gives 500 pulses for 1 rotation of the shaft, send 2000 to rotate motor shaft by one full rotation. The device takes between 5 - 10 ms to respond to this command.

Parameters:

position Absolute position as a 32 bit 2's compliment number. Set velocity limit bit in the function register (MMFunc_velocityLimit) and load velocity limit register (mmcreg_-velocityLimit) to limit average velocity during motion.

Returns:

0 on success, -1 on error.

4.5.3.5 int MotionMind::moveToRelative (int32_t *position*)

Specify movement to a position relative to current position, when operating in mode 4 (serial closed loop). Ensure that the PID filter is tuned, and note that the encoder is decoded on a 4:1 ratio (i.e., if the encoder gives 500 pulses for 1 rotation of the shaft, send 2000 to rotate motor shaft by one full rotation. The device takes between 5 - 10 ms to respond to this command.

Parameters:

position Relative position as a 32 bit 2's compliment number. Set velocity limit bit in the function register (MMFunc_velocityLimit) and load velocity limit register (MMReg_-velocityLimit) to limit average velocity during motion.

Returns:

0 on success, -1 on error.

4.5.3.6 int32_t MotionMind::readRegister (MMReg_t *reg*)

Read contents of a register. The returned values may have 1, 2 or 4 bytes of valid data, and may be in 2's compliment data format. See Section 6.0 (Register Definitions - Function/Status Bits) of the user manual for more informaiton. The device takes between 5 - 10 ms to respond to this command. To check whether a particular bit (example MMFunc_posPwUp) is set in the function register, use this method as follows

```
int16_t func = readRegister(MMReg_function);
func &= MMFunc_posPwUp;
if(func) {
cout << "Position will be restored from EEPROM on power up" << endl;
}
```

The status register can be read in the same manner.

Parameters:

reg The register index.

Returns:

register value. The return value is undefined if the call didn't successfully execute.

4.5.3.7 int MotionMind::reset ()

Stops the motor and does a software reset. Allow a few seconds before calling any other method after a call to this function.

Returns:

0 on success, -1 on error.

4.5.3.8 int MotionMind::restore ()

Restore factory defaults. The motor is automatically stopped after successful execution of this command NOTE: The device takes up to 40 ms to respond to this command.

Returns:
0 on success, -1 on error.

4.5.3.9 int MotionMind::writeRegister (MMReg_t *reg*, int32_t *val*)

This command can be used to set internal registers in any mode of operation. Register values set using this call are not retained after power cycling the controller. Use writeStoreRegister() to make permanent changes to register contents. The registers are 1, 2 or 4 bytes in length, and some of them expect 2's compliment data format. The user must take care to send properly formatted data to this call. The function and status registers (MMReg_function, MMReg_status) contain bit fields where each bit is related to a functionality. See Section 6.0 (Register Definitions - Function/Status Bits) of the user manual for more informaiton. The device takes between 5 - 10 ms to respond to this command. To enable a particular bit (example MMFunc_posPwUp) in the function register, use this method as follows

```
int16_t func = readRegister(MMReg_function);
writeRegister(MMReg_function, func | MMFunc_posPwUp);
```

In order to disable a functionality, the user code would look like the following

```
int16_t func = readRegister(MMReg_function);
writeRegister(MMReg_function, func & ~MMFunc_posPwUp);
```

Parameters:
reg The register index.

val the desired value.

Returns:
0 on success, -1 on error (including attempted write into a read-only register).

4.5.3.10 int MotionMind::writeStoreRegister (MMReg_t *reg*, int32_t *val*)

This command can be used to set internal registers in any mode of operation. Register values set using this call are retained after power cycling the controller, and hence, used to modify default settings. Use writeRegister() to make temporary changes to register contents. The registers are 1, 2 or 4 bytes in length, and some of them expect 2's compliment data format. The user must take care to send properly formatted data to this call. The function register (MMReg_function) contains bit fields where each bit is related to a functionality. See Section 6.0 (Register Definitions - Function/Status Bits) of the user manual for more informaiton. NOTE: The device takes up to 40 ms to respond to this command. Upon reception of a valid command, the motor is automatically stopped. The settings are made permanent by writing to the internal EEPROM which has a life time of about 1,000,000 write cycles. Avoid overuse of this function call.

Parameters:
reg The register index.

val array with desired value (1,2, or 4 bytes).

Returns:
0 on success, -1 on error (including attempted write into a read-only register).

The documentation for this class was generated from the following file:

- MotionMind.hpp

Chapter 5

Motion Mind DC Motor Controller 1.0 File Documentation

5.1 MMClientServer.hpp File Reference

```
#include "MotionMind.hpp"
#include "putils/ShMem.hpp"
#include "putils/StatusReport.hpp"
#include <time.h>
#include <signal.h>
#include <semaphore.h>
```

Classes

- struct MMCommand
- class MMClient

  A non-blocking client to MMServer.

- class MMServer

  Provides a server for motion mind controller.

Defines

- #define MMCHANGESPEED 0x01
- #define MMMOVETOABS 0x02
- #define MMMOVETOREL 0x03
- #define MMMOVEATVEL 0x04

5.1.1 Define Documentation

5.1.1.1 #define MMCHANGESPEED 0x01

5.1.1.2 #define MMMOVEATVEL 0x04

5.1.1.3 #define MMMOVETOABS 0x02

5.1.1.4 #define MMMOVETOREL 0x03

5.2 MotionMind.hpp File Reference

```
#include <inttypes.h>
#include <stdlib.h>
```

Classes

- struct MMData
- class MotionMind

  A driver for the Solutions Cubed Motion Mind DC motor controller Revision 4.

Typedefs

- typedef enum MMReg MMReg_t
- typedef enum MMFunc MMFunc_t
- typedef enum MMStatus MMStatus_t
- typedef MMData MMData_t

Enumerations

- enum MMReg {

  MMReg_position = 0, MMReg_velocityLimit, MMReg_velocityFf, MMReg_function,

  MMReg_pTerm, MMReg_iTerm, MMReg_dTerm, MMReg_address,

  MMReg_pidScalar, MMReg_timer, MMReg_rcMax, MMReg_rcMin,

  MMReg_rcBand, MMReg_rcCount, MMReg_velocity, MMReg_time,

  MMReg_status, MMReg_revision, MMReg_mode, MMReg_analogCon,

  MMReg_analogFbck, MMReg_pwmOut, MMReg_indexPos }
- enum MMFunc {

  MMFunc_posPwUp = 0x0001, MMFunc_satProt = 0x0010, MMFunc_savePos = 0x0020, MMFunc_velLimit = 0x0040,

  MMFunc_activeStop = 0x0080, MMFunc_lastRc = 0x0100, MMFunc_adStep = 0x0200, MMFunc_adSerial = 0x0400,

  MMFunc_enableDb = 0x0800, MMFunc_selectFbck = 0x1000, MMFunc_virtLimit = 0x2000 }
- enum MMStatus {

  MMStatus_negLimit = 0x1, MMStatus_posLimit = 0x02, MMStatus_brake = 0x04, MMStatus_index = 0x08,

  MMStatus_badRc = 0x10, MMStatus_vnLimit = 0x20, MMStatus_vpLimit = 0x40 }

5.2.1 Typedef Documentation

5.2.1.1 typedef struct MMData MMData_t

5.2.1.2 typedef enum MMFunc MMFunc_t

5.2.1.3 typedef enum MMReg MMReg_t

5.2.1.4 typedef enum MMStatus MMStatus_t

5.2.2 Enumeration Type Documentation

5.2.2.1 enum MMFunc

Enumerator:

MMFunc_posPwUp
MMFunc_satProt
MMFunc_savePos
MMFunc_velLimit
MMFunc_activeStop
MMFunc_lastRc
MMFunc_adStep
MMFunc_adSerial
MMFunc_enableDb
MMFunc_selectFbck
MMFunc_virtLimit

5.2.2.2 enum MMReg

Enumerator:

MMReg_position
MMReg_velocityLimit
MMReg_velocityFf
MMReg_function
MMReg_pTerm
MMReg_iTerm
MMReg_dTerm
MMReg_address
MMReg_pidScalar
MMReg_timer
MMReg_rcMax
MMReg_rcMin
MMReg_rcBand
MMReg_rcCount
MMReg_velocity

MMReg_time
MMReg_status
MMReg_revision
MMReg_mode
MMReg_analogCon
MMReg_analogFbck
MMReg_pwmOut
MMReg_indexPos

5.2.2.3 enum MMStatus

Enumerator:

MMStatus_negLimit
MMStatus_posLimit
MMStatus_brake
MMStatus_index
MMStatus_badRc
MMStatus_vnLimit
MMStatus_vpLimit

Index