

2 Using C-Motion

C-Motion is a “C” source code library that contains code for communicating to the motion processor using a parallel or serial interface. Please contact PMD for the latest version of the C-Motion library.

All C-Motion examples are intended for use on a PC running a Windows® Operating System. The library also contains files which are MicroSoft Visual Studio® 6.0 projects. Every example folder will have a file with a “.dsp” extension. If Visual Studio 6.0 is installed on the PC, then clicking on the “.dsp” file will open the project and the user should be able to compile the project without errors. If a development environment besides Visual Studio is being used then C-Motion can still be utilized however the user is now responsible for setting up the project so that all of the necessary source, object, and library files are properly linked during a compile.

The developer’s initial utilization of C-Motion is intended for use with PMD’s Development Kit boards, which are inserted into either an ISA or PCI slot on a PC. Hence there are examples that are specific to ISA or PCI communication. However C-Motion is also useful to developers who have purchased a chip level product from PMD. In particular it is a useful starting point for developing the section of code that will run on their host processor that is responsible for communicating to the PMD chip.

One very important function of C-Motion is to translate one user level function call into several low level read and write instructions. These low level instructions, which are passed to the communication bus, are in the format of packet structures defined in the *Programmer’s Reference* or *Programmer’s Command Reference* for the product of choice. The theory is that the developer will only need to modify the low-level communication routines in C-Motion based on the type of host processor and the subsequent communication routines provided by that host processor.

C-Motion includes the following features:

- Axis virtualization
- Communicate to multiple motion processors
- Easily linked to any “C/C++” application
- Supports 16/16, 8/16 and 8/8 parallel communication modes
(8/8 mode not supported on Magellan)
- Support serial communication
- Supports Windows driver communication mode

The following files make up the C-Motion source code distribution:

C-Motion.h/C-Motion.c	Definition/declaration of the PMD Navigator command set
PMDpar.h/PMDpar.c	Parallel interface functions
PMDW32ser.h/PMDW32ser.c	Windows serial communication interface functions
PMDdrv.h/PMDdrv.c	Windows driver communication interface functions
PMDutil.h/PMDutil.c	General utility functions
PMDtrans.h/PMDtrans.c	Generic transport (interface) functions

PMDdecode.h	Defines the PMD and C-Motion error codes
PMDocode.h	Defines the control codes for Navigator commands
PMDtypes.h	Defines the basic types required by C-Motion
PMDdiag.h	Defines a string for each control code

C-Motion can be linked to your application code by including the above “C” source files in your application. Then, for any application source file that requires access to a PMD motion processor # include “C-Motion.h”.

By customizing the base interface functions in PMDpar.c or PMDW32ser.c, C-Motion can be ported to virtually any hardware platform. Use of C-Motion on generic platforms is covered in Section 3.2.2.2.

2.1 Theory of Use

C-Motion is a set of functions that encapsulate the PMD motion processor command set. The first parameter of every command is an “axis handle”. The axis handle is a structure containing information about the interface to the motion processor and the axis number that the handle represents. Before communicating to the motion processor, the axis handle must be initialized using the following sequence of commands:

```
// the axis handles
PMDAxisHandle hAxis1, hAxis2;
// open interface to PMD processor and initialize handle to axis one
PMDSetupAxisInterface_Parallel( &hAxis1, PMDAxis1, 0x340 );
// initialize handle to the second axis
PMDCopyAxisInterface( &hAxis2, &hAxis1, PMDAxis2 );
```

The above is an example of initializing communication using the parallel communication interface. PMDutil.c contains examples of initializing all of the available communication modes.

Once the axis handle has been initialized, any of the motion processor commands can be executed. C-Motion.h includes the prototypes for all motion processor commands as implemented in C-Motion. Refer to this file for the required parameters for each command. The *Programmer’s Reference* for the PMD product you are using is the primary source for information about the operation and purpose of each command.

As part of a normal startup procedure, the motion processor is often reset. This command only needs to be executed ONCE for each motion processor, and not for every axis. The following code demonstrates a chip reset using C-Motion.

```
PMDuint16 result;
PMDuint16 status;

// reset the PMD chip set that this axis resides on
// if more than one chip set is present, all of them should be
// reset here
result = PMDReset(&hAxis1);
```

```

// in the serial interface mode if an error occurred it is returned
// immediately with no need to call GetHostIOError, so in this
// code we check for any error before continuing.
// with the parallel interface the result code will always be
// PMD_ERR_CommandError since that bit is set whenever a reset
// occurs. If it ISN'T set then there is some other error
if ( (result != PMD_ERR_ChipsetReset) && (result != PMD_ERR_CommandError) )
{
    printf("Error: %s\n", PMDGetErrorMessage(result));
    return 0;
}

// after the reset the chip will be in the PMDChipsetReset state
result = PMDGetHostIOError(&hAxis1, &status);

// the above command should execute without error but we need to check
if ( (result != PMD_ERR_OK) || (status != PMD_ERR_ChipsetReset) )
{
    printf("Error: %s\n", PMDGetErrorMessage(result));
    return 0;
}

```

Every command returns a status code of type PMDuint16. The return code for every command executed should be checked before attempting to execute more commands.

```

PMDuint16 result,status;
result = PMDUpdate(&hXAxis);
if (result != PMD_ERR_OK)
{
    status = result;
    if (result == PMD_ERR_CommandError)
        PMDGetHostIOError( &hXAxis, &status );
    printf("Error: %s\n", PMDGetErrorMessage(status));
    return;
}

```

Internally, C-Motion checks the status of the HostIOError bit after issuing a command. If this bit is set it will return **PMD_ERR_CommandError**. The application should then call **PMDGetHostIOError** to clear the error bit and determine the cause of the error.

Many commands require additional parameters. Some standard values are defined by C-Motion and can be used with the appropriate commands. Refer to PMDtypes.h for a complete list of defined types. An example is shown below.

```

PMDSetBreakpoint(&hAxis1, PMDBreakpoint1, PMDAxis2,
PMDBreakpointActionAbruptStop, PMDBreakpointActualPositionCrossed);

```

The following functions are provided by C-Motion in addition to the PMD command set:

```
PMDuint16 PMDGetStatus(PMDAxisHandle* axis_handle);  
    returns the result of executing an IO status read operation  
PMDuint16 PMDHasError(PMDAxisHandle* axis_handle);  
    returns 1 if the error bit is set in the IO status word and returns 0 if it is not set.  
PMDuint16 PMDIsReady(PMDAxisHandle* axis_handle);  
    returns 1 if the ready bit is set in the IO status word and returns 0 if it is not set.  
PMDuint16 PMDHasInterrupt(PMDAxisHandle* axis_handle);  
    returns 1 if the interrupt bit is set in the IO status word and returns 0 if it is not set.  
void PMDCloseAxisInterface(PMDAxisHandle* axis_handle);  
    should be called to terminate an interface connection  
char *PMDGetErrorMessage(PMDuint16 errorCode);  
    returns a character string representation of the corresponding PMD Navigator or C-  
    Motion error code.  
void GetCMotionVersion(PMDuint8* MajorVersion, PMDuint8* MinorVersion);  
    returns the major and minor version number of C-Motion.
```

2.2 Non-Microsoft Compiling

As mentioned before C-Motion, as received, is fully compatible with Microsoft Visual Studio. When integrating C-Motion into code that will run on an embedded processor, it is very possible that, due to the type of embedded processor or OS being used, the developer will no longer be using a Microsoft compiler. As a result compile errors may be seen and some possible solutions will be discussed. (Note that this section does not attempt to address every possible problem that could arise with using a non-Microsoft compiler.)

C-Motion #includes a few Microsoft specific header files including CONIO.H, MEMORY.H, TIMEB.H and DOS.H. There are also functions related to the ISA driver that are Microsoft specific like **outp**, **outpw**, **inp** and **inpw**. Also the function **memset** may need replacing. Replacing these functions with equivalent functions specific to the platform being used will be necessary. This requirement will be addressed in greater detail in the subsequent chapter.

One compiler used for 8051 processors is uVision by Keil Software®, Inc. Several issues that may arise when using this and other similar compilers will be addressed here.

- The words “data” and “bit” are keywords and cannot be used as a variable name.

Example:

```
extern PMDuint16 PMDWriteBuffer(PMDAxisHandle* axis_handle, PMDuint16  
bufferID, PMDint32 data);
```

To compile this in uVision the definition must be changed to:

```
extern PMDuint16 PMDWriteBuffer(PMDAxisHandle* axis_handle, PMDuint16  
bufferID, PMDint32 data32);
```

and:

```
PMDuint16 PMDSetTraceStart(PMDAxisHandle* axis_handle, PMDAxis
traceAxis, PMDuint8 condition, PMDuint8 bit, PMDuint8 state);
```

Must be change to:

```
PMDuint16 PMDSetTraceStart(PMDAxisHandle* axis_handle, PMDAxis
traceAxis, PMDuint8 condition, PMDuint8 bit8, PMDuint8 state);
```

- No forward declarations are permitted.

Example from PMDPar.h:

```
typedef struct tagPMDIOTransport PMDIOTransport;
typedef struct tagPMDAxisHandle PMDAxisHandle;
```

uVision will not allow these statements here because **tagPMDIOTransport** and **tagPMDAxisHandle** have not been defined yet.