PMC Precision MicroControl Corp.

TechNOTE

Title: Interfacing to MCCL Macros from Windows Programs
Products(s): All
Keywords: Macros, MCAPI, Interfacing, User Registers
ID#: TN1025
Date: April 9, 1999

Summary
Many times it desirable to take advantage of the programmability of PMC's motion controllers from within a Windows program. This is particularly true for time critical tasks that need to be processed independent of the Windows Operating System, which is not a real-time system.

More Information
The key to interfacing Windows programs developed using the Motion Control API (MCAPI) with macro programs that run on the motion controller (written in Motion Control Command Language – MCCL) is to pass information using the user registers. Windows programs must use the general purpose registers on the motion controller to pass values to and from the MCCL macro. Registers are also used to inform the PC when the macro has completed.

As an example, let's construct a macro that will average a variable number of analog input readings (using one of the built in analog input channels on the motion controller. The macro will accept as arguments the channel number, number of readings, and an amount of time to wait between readings (dwell time). This example assumes an advanced motion controller, such as the DCX-AT200, that is capable of multi-tasking and floating point math. The registers used be the sample are as follows:

```
Argument Registers
Reg50 - A/D channel number
Reg51 - dwell between readings
Reg52 - number of readings (macro sets to zero when done)
Reg53 – results (average of all analog readings

Registers used internally by macros
Reg01 – sum of readings
Reg02 - number of readings – 1
```

The actual MCCL code is split into three macros. The startup macro (Macro #40) initializes the local variables and jumps to the main macro. The main macro (Macro #41) performs the data acquisition loop and jumps to the finish macro. Finally the finish macro (Macro #42) calculates the average of the readings and signals the PC that the macros have finished.

The initialization macro first zeros **Reg_{01}**, which will be used to hold the sum of the analog input readings (AL0,AR1). Then it subtracts one from the number of readings and saves this value in **Reg_{02}** for the repeat command "RP" used in the main macro (RA52,AS1,AR2). Finally it jumps to the main macro (MJ41).

The main macro reads an analog value using the channel number contained in **Reg$_{50}$** (GA@50). The sum of all previous readings is then added to this value and the result is returned to **Reg$_{01}$** (AA@1,AR1). Next the macro is put to sleep for a variable dwell period (WA@51). This process is repeated n – 1 more times, where n is the total number of analog readings (RP@2). At the end of the repeat sequence it jumps to the completion macro (MJ42).

The cleanup macro multiplies the accumulated sum by 1.0 (to convert the value to a floating point number), moves it to the accumulator, and divides by the number of readings (AL1.0,AM@1,AD@52). The result is copied to **Reg$_{53}$**, and **Reg$_{52}$** is zeroed to indicate that we have finished.

The Windows program opens the controller in ASCII mode so it can download macros to the controller. It first clears out any existing macros, then downloads our sample macros. The controller is then set to binary mode (faster, more robust) for the remainder of the program. Note that the macros only need to be downloaded once, at the beginning of the program.

```
hCtlr = MCOpen( 0, MC_OPEN_ASCII, NULL);
MCBlockBegin( hCtlr, MC_RESETM, NULL );         // clear old macros
pmcputs( hCtlr, “MD40,AL0,AR1,RA52,AS1,AR2,MJ41\r” );
pmcputs( hCtlr, “MD41,GA@50,AA@1,AR1,WA@51,RP@2,MJ42\r” );
pmcputs( hCtlr, “MD42,AL1.0,AM@1,AD@52,AR53,AL0,AR52\r” );
MCReopen( hCtlr, MC_OPEN_BINARY );
```

Next we need to “set-up” the registers from a windows program prior to executing the macros. To do this we use the **MCSetRegister( )** function:

```
channel = 1;                    // analog channel number
dwell_msec = 0.005;             // dwell time in seconds
nreadings = 1000;               // number of readings to average
MCSetRegister( hCtlr, 50, &channel, MC_TYPE_LONG );
MCSetRegister( hCtlr, 51, &dwell_msec, MC_TYPE_DOUBLE );
MCSetRegister( hCtlr, 52, &nreadings, MC_TYPE_LONG );
```

Finally, we start the macro and wait for **Reg$_{52}$** to go to zero before reading the results:

```
MCBlockBegin( hCtlr, MC_BLOCK_TASK, 40 );
MCBlockEnd( hCtlr, NULL );
do {
      MCGetRegister( hCtlr, 52, &done, MC_TYPE_LONG );
} while (done != 0 );
MCGetRegister( hCtlr, 53, &results, MC_TYPE_DOUBLE );
```