PMAC Throughput Testing Results

Research has been conducted to determine the rate at which PMAC can process a motion program. This document is intended to give a PMAC user a general idea as to how fast a motion program can be processed based on the data attained and provide the steps that were used to determine this rate. In addition, this document explains the various boundaries that are reached when pushing PMAC to its computational limit and has suggestions to further increase the throughput. A section devoted to PMAC NC for Windows is also included to explain throughput results attained for this application.

Throughput Fundamentals

The rate at which PMAC processes a motion program is specified in units of blocks per second, bps. A block is considered to be one line in the motion program regardless of how many axis are in the line. For example, X10, X10Y20, X10Y20Z2, TA1, etc. are all considered one block.

A bit of categorizing is in order. As far as PMAC is concerned, motion programs are of two types, external and internal. Internal programs reside in a standard program buffer within PMAC's memory. External programs, programs generated and transferred by a host computer, ultimately end up in one of PMAC's internal rotary buffers either in an ASCII format, or binary (binary only possible if using the Dual-Ported Rams (DPR) Binary Rotary Buffer). Furthermore, external binary programs may be transferred from host to PMAC in two ways, with or without preprocessing the ASCII program file. That is, when sending PMAC a program via the DPR binary rotary buffer, the host may first convert the PMAC ASCII motion program file to a PMAC binary file and then send the binary file piece meal to the DPR. Alternatively, the host can convert and download each line of the ASCII file on the fly one line at a time.

All three forms, internal and both external, were tested under a variety of servo frequencies, phase frequencies, master clock frequencies, number of axis in a single block, and also with and without streamlining PMAC's encoder conversion table. Appendices B-D show data (and bar charts) going from the most optimal PMAC settings, to the default factory settings respectively using DOS programs which used SPLINE mode, 1ms moves. Appendix E has data and bar charts illustrating the maximum throughput for Circular moves with optimal PMAC settings. Finally, Appendix F has NC for Windows data and charts for splined 1ms moves.

Splined and circular moves were chosen since every move statement in the motion program is processed. Whereas when PMAC is in Linear mode it may skip blocks, depending on whether or not it has enough time to calculate the move. As a result, using Linear move mode would lead to erroneous values for the PMAC throughput.

PMAC Computational Limits

What was found to limit the maximum throughput is discussed here. Recognizing what is keeping PMAC from running faster is important in determining what actions, if any, may be taken to remove the bottleneck. Keep in mind that none of the symptoms described below occurs, the host computer may be the slowest link in the chain. That is, for external programs, the rate at which the host can transfer the moves to PMAC is lower than the rate at which PMAC can process the moves.

For any given PMAC configuration, the limit on the throughput may be due to one of following:

Symptom	**Throughput Bottleneck**	**Remedies***
Watchdog trips card shuts down (and the tail stops wagging)	Not enough background time to keep the watchdog timer happy	See Table 2
A run time error occurs. Program run status bit is TRUE. Motors move at the last commanded velocity (run-away). Program runs fine, but can't increase feedrate	Not enough time to perform the motion calculations in one real time interrupt period. The maximum feedrate override is restricted by the servo interrupt frequency	Decrease I8. Also, see Table 2. Increase servo interrupt frequency
*See Appendix A for details on making optimizations		

A system that does not have enough background time will have the watch dog timer trip causing the card to shut down. Background time is the time left over after higher priority tasks have been done (i.e. Phasing, Closing servo loops, Motion planning, PLC0, PLCC0 etc.). Not having enough background time is a condition more frequently encountered with the lower speed 20Mhz PMAC than the 40/60 Mhz PMAC.

A Run Time Error occurs when the time at which the calculations for the next move are completed occurs after the move should have began. At every Real Time Interrupt PMAC does motion planning, if it is required, then PLC0 and finally PLCC0. If this does not occur often enough, the just described condition occurs. It is for this reason that I8, Run Time Interrupt period, for all test was set to 0. This mandates that Real Time Interrupts, RTI, occur at every servo cycle. No PLC0 /PLCC0 were used in the testing therefore the only thing done at RTI was to do if motion planning if needed. This is the optimum setting for I8 when trying to optimize throughput, since no significant processor time is taken unless it is required for the trajectory calculations.

When PMAC's throughput has been restricted by the feedrate override, increasing the servo interrupt frequency will increase throughput. The maximum feedrate override value can be determined by the following equation:

*Max_Feedrate_Override = 225 * Servo_Interrupt_Frequency/2.26(KHz)*

This condition, for example, usually occurred for the faster 40/60 MHz PMAC boards with the default servo interrupt frequency of 2.2 KHz. In this scenario, PMAC has computational power to spare, yet since the motion program can only have a feedrate override of 225% and the minimum move time is 1ms, throughput is restricted to 2250 blocks per second.

How to Increase Throughput

Before reading the conclusions deduced from the data it is suggested that the reader review PMAC's computational priorities, see Computational Features in the PMAC User Manual (User Guide). Once a good understanding of how PMAC performs its multitasking is achieved, the conclusions below will be better understood.

The following conclusion can be made from the data. In order of most significant impact, a higher throughput can always be attained by:

Optimization	**Throughput Increase**
Increasing the Master clock frequency	100-800%
Decreasing the number the axis per block	200-700%
Pre-process the ASCII file to a binary one.	80-100%
[1]Removing any unused entries from the Encoder Conversion Table	10 - 30%
Using an internal buffer, next fastest external binary, slowest external ASCII	10-15%
[1]Setting the Phase frequency equal to the servo frequency	5- 10%
[2]Change uncompiled PLCs to compiled PLCs	Floating point operations run 2-3 time faster and fixed point operations run 20-30 times faster.

See Appendix A for detailed information on making selected optimizations.

Trends:

The following trends have been found. Here we discuss the effects on throughput as a result of varying one parameter as all others remain fixed trajectory calculations (Run time error occurs).

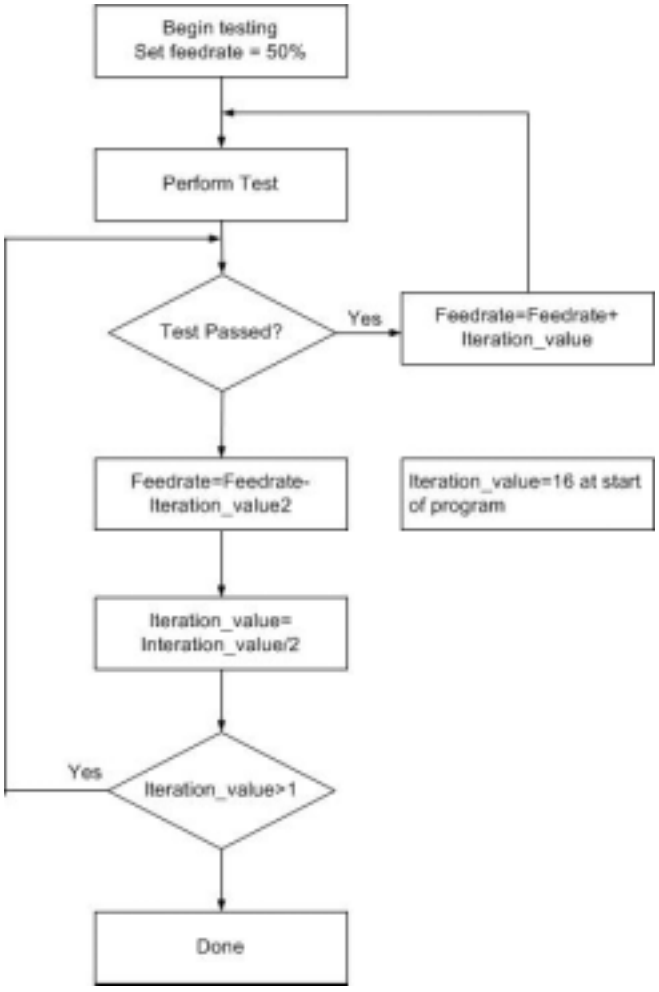
Increasing Servo Frequency:
As one increases the servo interrupt frequency PMAC spends more time on:

1. Sampling feedback devices, converting data (via the Encoder Conversion Table) and computing the output based on the difference in the commanded Vs actual position
2. Foreground tasks
   a. Motion planning
   b. PLCs and/or PLCCs (none of which were used in this analysis) and therefore less time becomes available for background tasks.

How the Data was attained

The basic methodology in attaining the data is first discussed followed by exactly how PMAC was configured for the various tests that were conducted.

The data was attained by developing a DOS based application, benchmrk.c (Available with ComLib). For any given test, the card was configured appropriately in both hardware (servo_frequency and/or Master clock frequency) as well as software (I-variables). Three functions were developed; all dedicated testing routines for each of the three methods of running a PMAC program. The main program would iteratively call one of these dedicated functions specifying the feedrate at which to run the PMAC program, and the function would return the results of the test (percent error) when completed. Depending on whether or not the last iteration ran successfully, the feedrate would be varied according to the algorithm illustrated below: (Keep in mind the move times were set to 1 ms)



All necessary conditions for a successful test run are listed and then described below:

1. Calculated run time was within 1% of the actual run time
2. A run time error, RTI, did not occur.
3. The watchdog timer did not trip.
4. A servo interrupt error did not occur

The first mode of failure, extended actual run time, occurs in both the external program types. The time that the program should take to execute (in milliseconds) is calculated by the following formula:

run_time_calc = (NUM_LINES+2)(100.0/feedrate); 2 extra TA added in spline*

If PMAC takes 1% longer to run the program than the calculated value a failure is considered to have occurred. Here the bottleneck is the rate at which PMAC can transfer data from the host (not enough background time), or if the host computer is slow, the rate at which the host can transfer moves to PMAC. In both cases, the PMAC rotary buffer at times may not have any moves in it and extra time will be spent decelerating to stop, possibly waiting for the next move and finally accelerating to speed.

A run time error occurs when the time at which the calculations for a move have completed is after the move should have started. When this happens PMAC will have all motors continue to move at the last commanded velocity and the program never finishes. In this case, a time-out counter in the testing algorithm determines that a run time error has occurred.

A watch dog timer will trip when insufficient background time was available for house keeping or the RTI is not occurring frequently enough (won't be the cause of failure here) or if the digital voltage goes below 4.5V (This too is not a factor in the tests performed).

Servo errors will occur when not all the calculations necessary for a single servo cycle can be completed in a single servo period. In this case, the PMAC will report a servo interrupt error in the appropriate coordinate systems status word. Nothing catastrophic occurs when this error is encountered. PMAC will continue any unfinished servo calculations in the following servo interrupt. The user will notice that the program will continue to run at exactly half the speed it should since every other servo cycle is now skipped.

Timing of Program Execution

In all tests the means of timing was by use of a high resolution (+/- 50 microseconds) PC based timer. The timer was started at the beginning of program execution and stopped when the last line of code was executed. For the internal ASCII program the beginning of program execution was considered to be the time immediately following the acknowledge character received by the host after sending PMAC a run command. The lag time between when PMAC actually begins running the program and when the PC timer starts timing is considered negligible. This is based on the fact that all tests used 3000 blocks of program code which resulted in a smallest run time of 670 ms. In comparison the maximum response time of issuing a run command to PMAC is under 3.5 ms. For both external programs the start time was considered to be just before the first program line was sent to a rotary buffer. For all types, a program was considered finished when the last line of code, which was an m-variable assignment statement, occurred. The M-Variable was defined to point to DPR. The host program would poll this memory location until it was assigned, at which point the PC-timer would stop. Again, the lag time between program termination and the stopping of the PC timer is considered negligible.

Timing could have also be done using PMAC's servo counter. In this case, program lines would have to be added to capture the servo counter at the start and end of the program. The time would have been calculated as the difference between the ending and starting servo counter values times the servo interrupt period. Actually, this was done to compare timing techniques and it was found that the results consistently differed by less than 0.5%.

Setting up PMAC

The communication functions used in benchmrk.c were from ComLib, the Delta Tau PMAC C-function communication library. Communication over the bus and DPR were established. PMAC, for all tests, was configured by sending it the following command strings:

$$$* **	;Card set to factory default state
I58=1	;Communication to ASCII-DPR established
I10 = xxx	;Set to appropriate value, a function of servo_frequency
I187=1	;Set default Accel time
I188=0	;Set the default S-curve time
I8 = 0	;Set Real Time Interrupt frequency equal to the servo frequency
&1	;Setup motors and coordinate system
#1->1x	;Define all axis being used for the test
#2->1y	;And so on depending on the number of axis, x, y, z, u, v, w
i100=1	;Activate all motors being used for a given test
i200=1	;And so on depending on the number of axis
i111=0	;So that motors don't really have to be hooked up
i211=0	;And so on depending on the number of axis
i125=$12c000	;So that limits and amp faults do not have to be grounded in hardware
i225=$12c004	;And so on depending on the number of axis
#1j/	;Close the loop on all motors active for the given test
#2j/	;And so on depending on the number of axis

Now the details of each testing method are described in pseudo code. The nomenclature for the pseudo-code is shown below:

Typeface	Meaning
Bold	Strings sent to PMAC
Bold Italic	Pseudo-Code

Internal ASCII Procedure

The following procedure, determines how fast PMAC can run a program stored in a standard program buffer in ASCII form. The routine first downloads the program, begins execution and the timer, and then waits for the last statement in the program to execute before stopping the timer. The last statement in the routine sets a bit in DPR which is being polled.

1. Make the M-Variable definition to point in unused DPR:
   m1023->y:$D000,15,1 ; This location is in DPR's control panel, an unused location
   This serves as the end of program flag.
2. Make a far pointer to same location:
   pmac_all_done = (unsigned int *) MK_FP(0xD400,0);
   The code above is in C. pmac_all_done points to PC segment D400(hex) and offset 0.
3. Set pmac_all_done flag to 0.
   ***pmac_all_done = 0;**
4. Download program to PMAC
   Open prog1000
   Clear INC SPLINE1 TA01
   m1023 = 0 // Set program_done flag to 0
   do {
   if(number_of_axis == 2)
   x1y1
   else if (number_of_axis ==3)
   x1y1z1

```
.
.
endif
lines_down = lines_down+1;
} while (test>0 AND lines_down<NUM_LINES);
dwell0 //So that Ms are synchronous
m1023=1;
Close
```

5. Change feedrate
   ```
   %feedrate
   delay(2000);// Must put this delay in so feedrate override takes effect before run.
   ```
6. Begin execution of program and wait for acknowledge
   ```
   b1000r
   ```
7. Begin timer
   ```
   time1 = hrt_read();
   ```
8. Wait until program has completed running, or a timeout has occurred, Recall the last statement is an m-variable assignment which modifies the 15th bit of the word polled by the line below ***i=0;***
   ```
   A_TIMEOUT = 100000;
   while(!(*pmac_all_done & 0x8000) && !kbhit() && i < A_TIMEOUT) i++;
   time2 = hrt_read();
   ```
9. Timed out?
   ```
   if(i >= A_TIMEOUT){
   ```
   //A run time error has occurred. Exit procedure return failure condition ***return FALSE; }***
10. Determine how long it program took to execute, and whether or not a successful run.
    ```
    run_time_pc = (time2-time1)/(1193.0*1000.0);// PC CLOCK
    bps = (lines_down+3.0)/run_time_pc;
    run_time_calc = ((NUM_LINES+2.0)*.001)*(100.0/feedrate); // in seconds
    percent_error = ((run_time_pc-run_time_calc)/run_time_calc)*100.0;
    //  display run time calculated, actual, and error.
    ```
11. Return error
    ```
    if(error > 1)
    return FALSE
    else
    return TRUE;
    ```

External Binary Procedure

This procedure checks the rate at which a PMAC program may be downloaded to the DPR binary rotary buffer from ASCII code generated on the fly (loading from a file has been found to be the same speed).

It has been found that a 66Mhz DX2 PC clone has been able to transfer program blocks from the PC to the DPR Rotary buffer at the following rates:

#Axis	Transfer Rate (BPS)
1	8681
2	6700
3	5900
4	4500
6	3500

This is useful to know since if the BUS is not faster than PMAC's processing speed the host becomes the bottleneck in the transfer process.

PMAC program command processing rate: Determined by dividing the number of blocks sent by the difference in time from when the first block is sent and when the final block is sent. The last program line is a m-variable assignment. This M-Variable has been defined a m1023>y:$D000,15,1, a bit that is not being used in the DPR control panel area.

1. Initialize and enable the PMAC DPR Binary rotary buffer
   if (DProtBufInit(id,DPR_ROT_SIZE) < 0){
   //Report an error and return 1 RETURN 1;
   }
   DProtBuf(id,ON);
2. Define a rotary buffer internal to PMAC
   DEF ROT1500
3. Make M-Variable definition to point in unused DPR, this serves as the end of program flag:
   m1023->y:$D000,15,1 ;
4. Make a far pointer to same location:
   pmac_all_done = (unsigned int *) MK_FP(0xD400,0);
5. Set pmac_all_done flag to 0. pmac_all_done = 0;
   ****pmac_all_done = 0;***
6. Open PMAC's internal rotary buffer
   OPEN ROT
7. Change the feedrate override value.
   %feedrate
   delay(2000);// Must put this delay in so feedrate override takes effect before run.
8. Begin execution of rotary program
   "b1000r"
9. Begin download of program
   clear
   inc
   SPLINE1 TA01
   m1023=0 // For PC Timer
10. Start timer
    time1 = hrt_read();
11. Begin downloading NUM_LINES program lines

```
    timeout = 0;
    run_time_error_flag = FALSE;
top:
do {
    if(number_of_axis == 2)
        test = DPASCIIStrToRot("x1");
    else if (number_of_axis ==3)
        test = DPASCIIStrToRot("x1y1");
.
. }
if(test>0){ // Line sent to PMAC
    lines_down = lines_down + 1;
    timeout=0; // Reset timeout
```

```
}
else
    timeout++;// Buffer full, or PMAC has a run time error
} while (lines_down<(NUM_LINES));
if(lines_down < NUM_LINES){
    if( test != ERR_DPR_BUFFERBUSY)
        // Report to screen that Buffer has had an error, i.e. a bad command sent
else
        // Report that DPR rotary buffer has been filled
        if(timeout < THE_TIMEOUT) // Have not exceeded the timeout?
            goto top;
        else
            run_time_error_flag = TRUE;
```

12. Send the last line down

```
test=-1;
timeout=0L;
while(test < 0 AND timeout<THE_TIMEOUT){
    test = DPASCIIStrToRot(id,"m1023=1");// For PC timer
    timeout++;
}
if(timeout >= THE_TIMEOUT)
    return 2.0; // Program took to long for execution
```

13. Wait until program has completed running, last statement assigns m1023

```
i=0;
while(*pmac_all_done&0x8000 == 1 AND run_time_error_flag==0 AND
i<THE_TIMEOUT)
    i++;
time2 = hrt_read();
```

14. Close PMAC's rotary program buffer

```
CLOSE
```

15. Timed out? If yes, a run time error occurred.

```
if(run_time_error_flag || i >= THE_TIMEOUT){
    // Display an error message
    return 1.0;
}
```

16. Do necessary calculations,

```
run_time_pc = (time2-time1)/(1193.0*1000.0);
bps = (lines_down+2.0)/run_time_pc;
run_time_calc = ((NUM_LINES+2)*.001)*(100.0/feedrate)//2 for added TA times
error = ((run_time_pc-run_time_calc)/run_time_calc)*100.0;
```

17. Display results

```
Calculated run time = run_time_calc, Error = error
PMAC rotary program process rate = bps
```

18. Return Error

```
return error;
```

External ASCII Procedure

This procedure checks the rate at which a PMAC program may be downloaded to the PMAC internal rotary buffer when moves are transferred thru the DPR in ASCII form. Enhanced throughput may be attained using interrupt-based handshaking. Here, the DPR fixed background buffer takes some time in background to update.

1. Configure I16 and I17.
   I16=10
   I17=115
2. Enable DPR Background stuff.
   I49=1// Enable DPR background update **I159 = 1** // 1st motor/cs only
3. Define a rotary buffer internal to PMAC.
   DEF ROT1500 // Buffer size is dependant on number I17 and number of axis
   // A conservative value buf_size = num_axis*lines_ahead*1.5
4. Make M-Variable definition to point in unused DPR:
   m1023->y:$D000,15,1 //This serves as the end of program flag.
5. Make a far pointer to same location:
   pmac_all_done = (unsigned int *) MK_FP(0xD400,0);
6. Set pmac_all_done flag to 0.
   ****pmac_all_done = 0;***
7. Open the rotary buffer.
   OPEN ROT
8. Change feedrate override.
   %feedrate
   delay(2000);// Must put this in so feedrate override takes effect
9. Begin execution of rotary program
   "b1000r"
10. Begin download of program
    inc
    SPLINE1
    TA01
    m1023=0 // For PC Timer
11. Begin downloading NUM_LINES program lines.

```
    timeout = 0;
    run_time_error_flag = FALSE;
    time1 = hrt_read();
top:
    while(!DProtBufFull(id,1) AND lines_down<NUM_LINES)
    { if(number_of_axis == 2)
        X1
      else if (number_of_axis ==3)
      X1Y1
.
.
       endif lines_down++;
      }
     if(lines_down < NUM_LINES){
       // DPR rotary buffer has been filled
```

```
        if(DPsysRunTimeError(id,1)){
            //A run time error has occurred
            return 1.0;
        }
        goto top;
    }
    while(DProtBufFull(id,1)); // Wait while the rotary buffer is full
```

12. Send the last line of the program.
    M1023=1 // For PC timer
13. Wait until program has completed running, last statement assigns m1023
    i=0;
    while(!(*pmac_all_done & 0x8000) AND !run_time_error_flag)
        i++;
    time2 = hrt_read();
14. Close PMAC's rotary buffer.
    CLOSE
15. How long did it take according to the PC?
    run_time_pc = (time2-time1)/(1193.0*1000.0);
    bps = (lines_down+2.0)/run_time_pc;
    run_time_calc = ((NUM_LINES+2)*.001)*(100.0/feedrate);//2 for added TA times
    error = ((run_time_pc-run_time_calc)/run_time_calc)*100.0;
16. Display results.
    Calculated run time = run_time_calc, Error = error
    PMAC rotary program process rate = bps
17. Return Error.
    return error;

PMAC NC for Windows Throughput Testing

Delta Tau's PMAC NC for Windows, the first fully capable CNC controller providing an open-architecture approach to machine tools, was also tested for throughput (See Appendix F). This application, as with any application that has to display data real time and respond to a multitasking system on a continual basis, can be expected to have lower throughput than the dedicated, non-visual, DOS based testing that was described above.

The NC for Windows application uses only the external DPR binary rotary buffer for transferring motion programs to PMAC. The two modes in which this is done, each line converted to binary on the fly (ASCII->Binary) or preprocessed (Binary->Binary), were done for both 1 and 2 coordinate systems as well as the three PMAC master clock frequencies (20,40 & 60 MHz). Furthermore, two host computer speeds, 486-33 and 486-66 were evaluated. Since the PC is the bottleneck for throughput in nearly all cases (except for the 486-66, 20 MHz PMAC case), the faster the PC the higher the throughput.

Conditions for Testing

PMAC

The encoder conversion table and phase frequency were streamlined, I8 was set to 0, and two PLCs were running, PLC1 and PLC2. A file containing 10,000 SPLINE mode moves each of 1ms was used to conduct the test. The internal rotary buffer size was set to half that of the DPR rotary buffer size. More on buffer size is discussed below.

Host
Only two applications were running at the time of testing, the NC for Windows Executive program and CNC.EXE. Obviously, the more applications running the less time NC for Windows has to transfer the motion program.

The default rotary buffer sizes (set by *dprBinRot1Size* and *dprBinRot2Size)* that were used are shown below.

Number of Coordinate Systems	DPR Rotary buffer sizes (long integers 32-bit)	PMAC Internal Rotary Buffer Sizes (48-bit words)
1	2000	1000
2	1000	500

Timing of Program Execution

The following PLC was downloaded to PMAC for timing how long the motion program took to execute.

```
i5=2
m1000->x:$0,0,24,u
m1001->x:$0818,0,1
p1010=0

open plc1
clear if(m1001 = 1)
      if(p1010 = 0)
      p1001=m1000
      p1010=1
  endif
else
      if(p1010 = 1)
        p1002 = m1000
        p1010=0
        p1003=(p1002-p1001)*.000442
  endif
endif
close

i5=2 ENAPLC1
```

Optimizing SYS.INI file Parameters

Several parameters in the SYS.INI file, pertinent to throughput are now discussed.

dprBinRot1Size and **dprBinRot2Size**
The rotary buffer sizes in these tests play a more important role in determining maximum throughput than in the DOS based throughput testing applications. Why is this? Once the PMAC rotary motion program execution has begun, PMAC NC for Windows will update the display only when the DPR buffer has been filled or PMAC NC for Windows has downloaded *MaxBlkPerDisp* (a sys.ini parameter) number of lines. If the time it takes to update the display is longer than it takes PMAC to execute all the moves in the buffer then PMAC will bring all motors to a stop and wait for the next move (which appears after the host is done updating the display). If the condition just described occurs the run would be considered invalid, since a good run is considered to have all moves completely blended with no dwells.

MaxBlkPerDisp
If the parameter, *MaxBlkPerDisp*, exists in the SYS.INI initialization file, it should be set to as large as the DPR rotary buffer size. This parameter is used when converting RS274 to binary. The CNC.EXE application will convert and download until the DPR buffer has been filled or after MaxBlkPerDisp blocks have been processed

MaxFeedOvrd
MaxFeedOvrd restricts the maximum feedrate override value. A default value of 200% restricts the user to a maximum of 2000 blocks per second. Increase this value to 10 times the maximum number of blocks per second you wish to achieve.

Appendix A Making Optimizations

Removing Unused Encoder Conversion Table Entries

Removing unused entries in the encoder conversion table (ECT), and changing the phase frequency to the servo frequency are easily done. The ECT has nine entries in it by default. It assumes that there are eight motors with eight encoders as feedback, and the last entry will use encoder number 4 as a TIMEBASE. As an example to optimizing the ECT only using N motors, not eight, and not using the TIMEBASE entry. To remove the unused entries in the ECT, write a value of 0 to the ECT base address (1824 dec) plus N in the Y address space. If N = 4, then this would be accomplished with the on-line PMAC command WY:1828,0.

The phase frequency can be set to the servo frequency by changing hardware jumpers (E3-E6 and E 29-E33). For example to have a phase and servo frequency of 2.2 KHz, place jumpers on E3 through E6 and put a jumper on E29 leaving E30-E33 without a jumper. PMAC I-Variable, I10 (specifies time between servo interrupts) will not need to be changed from its default value in this case since 2.2KHz is the factory default servo frequency. Additional info on these jumpers is given below.

E3 - E6: Servo Clock Frequency Control

The servo clock (which determines how often the servo loop is closed) is derived from the phase clock (see E98, E29 - E33) through a divide-by-N counter. Jumpers E3 through E6 control this dividing function.

E3	E4	E5	E6	Servo Clock = Phase Clock Divided by N	Default and Physical Layout E3 E4 E5 E6
ON	ON	ON	ON	N = divided by 1	
OFF	ON	ON	ON	N = divided by 2	
ON	OFF	ON	ON	N = divided by 3	
OFF	OFF	ON	ON	N = divided by 4	Only E5 and E6 ON (20MHz)
ON	OFF	ON	ON	N = divided by 5	
OFF	ON	OFF	ON	N = divided by 6	Only E4 and E6 ON (30MHz)
ON	OFF	OFF	ON	N = divided by 7	
OFF	OFF	OFF	ON	N = divided by 8	
ON	ON	ON	OFF	N = divided by 9	
OFF	ON	ON	OFF	N = divided by 10	
ON	OFF	ON	OFF	N = divided by 11	
OFF	OFF	ON	OFF	N = divided by 12	
ON	ON	OFF	OFF	N = divided by 13	
OFF	ON	OFF	OFF	N = divided by 14	
ON	OFF	OFF	OFF	N = divided by 15	
OFF	OFF	OFF	OFF	N = divided by 16	

Note: The setting of I-Variable I10 should be adjusted to match the servo interrupt cycle time set by E98, E3 -- E6, E29 -- E33, and the master clock frequency. I10 holds the length of a servo interrupt cycle, scaled so that 8,388,608 equals one millisecond. Since I10 has a maximum value of 8,388,607, the servo interrupt cycle time should always be less than a millisecond (unless making the basic unit of time on PMAC something other than a millisecond). To have a servo sample time greater than one millisecond, the sampling may be slowed in software with variable Ix60.

Frequency can be checked on J4 pins 21 and 22.

Note:

If E40-E43 are set up so that the card has a software address other than @0, the servo clock signal must be received over the serial port from card @0, so these jumpers have no effect.

E29 - E33: Phase Clock Frequency Control

Jumpers E29 through E33 control the speed of the phase clock, and, indirectly, the servo clock, which is divided down from the phase clock (see E3 - E6). No more than one of these five jumpers may be on at a time.

E29	E30	E31	E32	E33	Phase Clock Frequency		Default and Physical Layout
					19.6608 MHz Master Clock	29.4912 MHz Master Clock	
ON	OFF	OFF	OFF	OFF	2.26 KHz	3.39 KHz	**E29**
OFF	ON	OFF	OFF	OFF	4.52 KHz	6.78 KHz	**E30**
OFF	OFF	ON	OFF	OFF	9.04 KHz	13.55 KHz	**E31**
OFF	OFF	OFF	ON	OFF	18.07 KHz	27.10 KHz	**E32**
OFF	OFF	OFF	OFF	ON	36.14 KHz	54.21 KHz	**E33**

Note: If jumper E98 has been changed to connect pins 2-3 (default is 1-2), the phase clock frequency is exactly 1/2 that shown in the above table.

Note: If E40-E43 are set so that the card has a software address other than @0, the phase clock signal must be received over the serial port from card @0, so these jumpers have no effect.

Appendix B Throughput Data, Encoder Conversion Table and Phase Frequency Minimized

Note:

The external binary tests for appendices B-E were done in the conversion on the fly mode. That is, the ASCII program file was not preprocessed before the test was conducted, rather each line of ASCII in the file was converted and downloaded as the motion program was running. Preprocessing the file can improve throughput significantly (as seen in the NC for Windows tests, up to 100%) when the host computer is the bottleneck. The test results shown in appendices B-E were done on a 486-66 PC with programs in DOS. With this combination, the host computer was never the slowest link in the process.

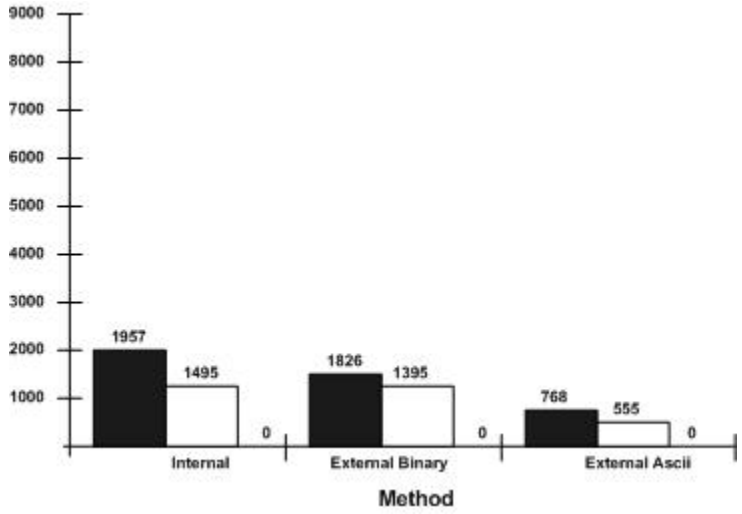
PMAC Setup: Factory default, V1.15a 2 AXIS Splined, E.C.T. and Phase Freq. Streamlined				
		Throughput (bps)		
Master Clock Freq (Mhz)	Servo Freq. (Khz)	Internal	External Binary	External ASCII
20	2.26	1957	1826	768
20	4.52	1495	1395	555
20	9.04	0	0	0
40	2.26	2248	2241	2241
40	4.52	4496	4496	2100
40	9.04	4356	4039	1626
60	2.26	2248	2250	2247
60	4.52	4497	4496	3375
60	9.04	7796	6958	2988

PMAC Setup: Factory default, V1.15a 3 AXIS Splined, E.C.T. and Phase Freq. Streamlined				
		Throughput (bps)		
Master Clock Freq (Mhz)	Servo Freq. (Khz)	Internal	External Binary	External ASCII
20	2.26	1294	1211	546
20	4.52	877	817	347
20	9.04	0	0	0
40	2.26	2241	2241	1827
40	4.52	3644	3404	1530
40	9.04	2522	2342	1002
60	2.26	2247	2241	2241
60	4.52	4497	4497	2571
60	9.04	4913	4489	2055

PMAC Setup: Factory default, V1.15a 4 AXIS Splined, E.C.T. and Phase Freq. Streamlined				
		Throughput (bps)		
Master Clock Freq (Mhz)	Servo Freq. (Khz)	Internal	External Binary	External ASCII
20	2.26	937	877	410
20	4.52	498	466	198
20	9.04	0	0	0
40	2.26	2250	2243	1429
40	4.52	2451	2433	1147
40	9.04	1485	1389	610
60	2.26	2246	2248	2241
60	4.52	4384	4198	2015

PMAC Setup: Factory default, V1.15a 6 AXIS Splined, E.C.T. and Phase Freq. Streamlined				
		Throughput (bps)		
Master Clock Freq (Mhz)	Servo Freq. (Khz)	Internal	External Binary	External ASCII
20	2.26	488	458	226
20	4.52	0	0	0
20	9.04	0	0	0
40	2.26	2001	1900	963
40	4.52	1435	1365	676
40	9.04	0	0	0
60	2.26	2242	2243	1569
60	4.52	2740	2569	1281
60	9.04	1635	1535	744

.15 A, Factory Default, 2-Axis Program, Master Clock Freq. 40 Mhz
ECT and Phase Frequency Minimized



Appendix C Throughput Data, Encoder Conversion Table Minimized

PMAC Setup: Factory default, V1.15a 2 AXIS		E.C.T. Streamlined		
		Throughput (bps)		
Master Clock Freq (Mhz)	Servo Freq. (Khz)	Internal	External Binary	External ASCII
20	2.26	1784	1674	695
20	4.52	1405	1310	516
20	9.04	0	0	0
40	2.26	2248	2241	2241
40	4.52	4480	4490	2066
40	9.04	4356	4039	1626
60	2.26	2248	2250	2247
60	4.52	4497	4480	3355
60	9.04	7696	6958	2988
PMAC Setup: Factory default, V1.15a 3 AXIS Splined, E.C.T. Streamlined				
		Throughput (bps)		
Master Clock Freq (Mhz)	Servo Freq. (Khz)	Internal	External Binary	External ASCII
20	2.26	1196	1096	506
20	4.52	79	734	312
20	9.04	0	0	0
40	2.26	2241	2241	1787
40	4.52	3554	3311	1499
40	9.04	2522	2342	1002
60	2.26	2247	224 7	2247
60	4.52	4497	4497	2522
60	9.04	4913	4489	2055

PMAC Setup: Factory default, V1.15A 4 AXIS Splined, E.C.T. Streamlined				
		Throughput (bps)		
Master Clock Freq (Mhz)	Servo Freq. (Khz)	Internal	External Binary	External ASCII
20	2.26	847	797	367
20	4.52	438	418	168
20	9.04	0	0	0
40	2.26	2241	2243	1390
40	4.52	2539	2373	1112
40	9.04	1485	1389	610
60	2.26	2247	2248	2241
60	4.52	4410	4137	1946
60	9.04	3379	3137	1450
PMAC Setup: Factory default, V1.15A 6 AXIS Splined, E.C.T. Streamlined				
		Throughput (bps)		
Master Clock Freq (Mhz)	Servo Freq. (Khz)	Internal	External Binary	External ASCII
20	2.26	428	408	199
20	4.52	0	0	0
20	9.04	0	0	0
40	2.26	1930	1826	930
40	4.52	1380	1320	655
40	9.04	0	0	0
60	2.26	2247	2247	1540
60	4.52	2679	2521	1270
60	9.04	1635	1535	744

Appendix D PMAC Throughput Results (No Optimizations)

The following results are from a PMAC, prom version 1.15A, which was not changed from factory default settings with the exception of enabling motors, setting I8 = 0.

PMAC Setup: Factory default, PROM V1.15a 2 AXIS Splined				
		Max. Throughput (bps)		
Master Clock Freq (Mhz)	Servo Freq. (Khz)	Internal	External Binary	External ASCII
20	2.26	1623	1515	606
20	4.52	1086	997	367
20	9.04	0	0	0
40	2.26	2241	2241	2164
40	4.52	4476	4476	1837
40	9.04	3449	3167	1201
60	2.26	2240	2234	2243
60	4.52	4497	4478	3107
60	9.04	4495	4477	2452

PMAC Setup: Factory default, PROM V1.15a 3 AXIS Splined				
		Throughput (bps)		
Master Clock Freq (Mhz)	Servo Freq. (Khz)	Internal	External Binary	External ASCII
20	2.26	1096	1032	466
20	4.52	598	555	219
20	9.04	0	0	0
40	2.26	2241	2241	1711
40	4.52	3300	3085	1400
40	9.04	1980	1840	780
60	2.26	2247	2241	2241
60	4.52	4476	4476	2460
60	9.04	4476	4170	1860

PMAC Setup: Factory default, PROM V1.15a 4 AXIS Splined				
		Throughput (bps)		
Master Clock Freq (Mhz)	Servo Freq. (Khz)	Internal	External Binary	External ASCII
20	2.26	767	718	340
20	4.52	319	0	0
20	9.04	0	0	0
40	2.26	2250	2243	1355
40	4.52	2300	2190	1052
40	9.04	1770	1062	470
60	2.26	2248	2247	2195
60	4.52	4260	4018	1920
60	9.04	2990	2820	1300

PMAC Setup: Factory default, PROM V1.15a 6 AXIS Splined				
		Throughput (bps)		
Master Clock Freq (Mhz)	Servo Freq. (Khz)	Internal	External Binary	External ASCII
20	2.26	400	370	177
20	4.52	0	0	0
20	9.04	0	0	0
40	2.26	1890	1795	920
40	4.52	1330	1256	625
40	9.04	0	0	0
60	2.26	2240	2243	1540
60	4.52	2240	2450	1246
60	9.04	1490	1390	680

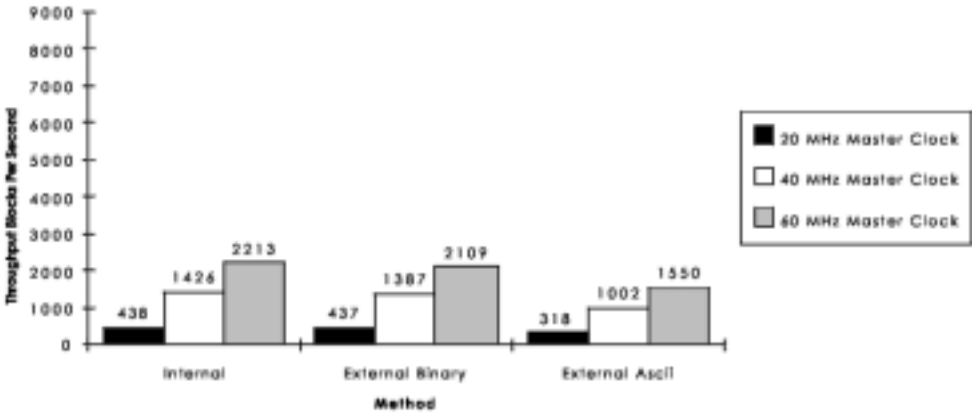
Appendix E PMAC Throughput Results (Circular Moves)

PMAC Setup: Factory default, V1.15a 2 AXIS Circle Moves, E.C.T. and Phase Freq. Streamlined SIF=2.2Khz			
Master Clock Freq (Mhz)	Internal	External Binary	External ASCII
20 MHz Master Clock	438	437	318
40 MHz Master Clock	1426	1387	1002
60 MHz Master Clock	2213	2109	1550

Note:

Servo Interrupt Frequencies greater than 2.2 kHz were not attempted because previous results have shown that this would not improve the throughput. Increasing the servo interrupt frequency only enhances throughput when the bottleneck is the feedrate override. In the 20 MHz case decreasing the servo interrupt frequency by one half will likely increase throughput although the servo performance will decline.

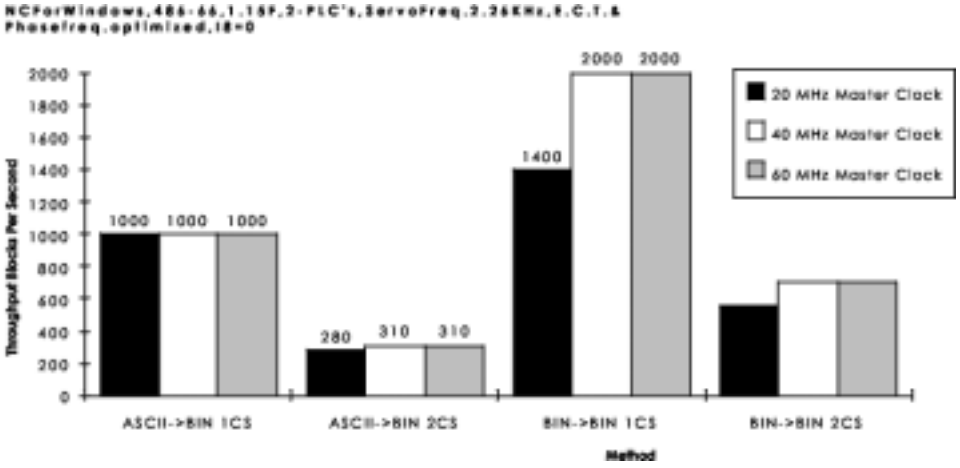
1.15A, 2-Axis Circular (I13 = TA = TM = 1m s), E. C. T. and Phase Freq. Minimized



Appendix F NC for Windows Throughput Results (Spline Moves)

The NC for Windows application uses only the external DPR binary rotary buffer for transferring motion programs to PMAC. The two modes in which this is done, each line converted to binary on the fly (ASCII->Binary) or preprocessed (Binary->Binary), are shown for both 1 and 2 coordinate systems as well as the three PMAC master clock frequencies. The difference in the two data sets below is only the speed of the host computer, 486-33 and 486-66. Since the PC is the bottleneck for throughput in all cases (except for the 486-66, 20 MHz PMAC case), the faster the PC the higher the throughput.

PMAC Setup: Factory default, V1.15F 2 AXIS Spline Moves, E.C.T. and Phase Freq. Streamlined, SIF=2.2Khz,486-33, 2-PLCs				
Master Clock Freq (Mhz)	ASCII->BIN 1CS	ASCII->BIN 2CS	BIN->BIN 1CS	BIN->BIN 2CS
20 MHz Master Clock	300	120	1100	350
40 MHz Master Clock	300	120	1100	350
60 MHz Master Clock	300	120	1100	350



PMAC Setup: Factory default, V1.15F 2 AXIS Spline Moves, E.C.T. and Phase Freq. Streamlined, SIF=2.2Khz,486-66, 2-PLCs				
Master Clock Freq (Mhz)	ASCII->BIN 1CS	ASCII->BIN 2CS	BIN->BIN 1CS	BIN->BIN 2CS
20 MHz Master Clk	1000	280	1400	550
40 MHz Master Clk	1000	310	2000	700
60 MHz Master Clk	1000	310	2000	700