Using PMAC in a Learn Mode

There are a few ways to accomplish the task of having PMAC learn a series of motor positions. The most general and usually the most useful involve having the PMAC connected to a host computer. This paper will focus on such applications.

In a learn mode, some sort of signal must be given to trigger the capturing of the motor positions. This trigger can be based on time, user input (a keystroke or push-button), motor position (using PMAC’s position capture ability), or some other event. The trigger type and its frequency will be one of the major factors that need to be considered when determining how to read the motor position information from PMAC. The following discussion will be based on a 20Mhz PMAC with a 2.2kHz servo for 4-axes and a 1.5msec Real Time Interrupt (RTI).

From the host programmers point of view, one of the easiest methods to get the motor positions is to have the host query PMAC for the information and then store these values in a file on the host. This can be done using any PMAC communications method (Serial, Bus, or DPRAM). The deciding factors for choosing between these methods is the trigger frequency and PMAC’s response time to a position query. Over the PC bus PMAC can respond to a position data query within 2msec, but the communication speed depends upon PLC and motion program duty cycle, number of servoed axes, and a few other processes. An average response time is in the area of 15msec, but it could be shorter or longer depending on what PMAC is doing. The busier PMAC is with PLC programs and motion program execution, the slower this response can be. In most learn mode applications, the communications delay is not a problem because the axes are moved to a location, stopped, then the trigger is given to grab the motor positions. For these types of applications choosing a communications mode will be most likely affected by other system factors and not by the learn mode. All of the above mentioned communications mode will be able to handle the trigger frequency of this type of learn mode.

In applications where the axes are not stopped before the trigger is given, the communications delay might cause a problem. For these applications, the following factors must be considered. Can the communications method perform at the desired rate? For instance, a 5msec trigger rate over the serial port at 9600 Baud cannot be processed. Also, will PMAC’s response delay to a query cause a problem. When PMAC is queried for information, the response time to that query is not fixed. It is easy to say that PMAC will respond within a given time period but it is impossible to say that the response will take exactly a given amount of time. Even if the host queries at a fixed rate, it is not guaranteed that PMAC responds in the same amount of time to each query. An average fluctuation in response time is around 5msec, but if the duty cycle changes so can this response delay. If the trigger is coming every 100msec, a 5msec response delay is probably not a problem. If the trigger is coming every 10msec, a 5msec response delay is probably a problem.

If the response delay is not acceptable for a given application, then a DPRAM communications method must be used. The DPRAM can be set up so that PMAC updates the positions at a fixed rate automatically. With this method the data will always be ready for the host, there is no communications delay and the PLC and motion program load does not matter. The feature that accomplishes this is called the Servo Fixed Data Buffer and it is described in the May 1994 version of the PMAC User’s Manual Addendum for PROM Versions 1.14x and 1.15x. The potential problem here is the speed of the host. For example, assume that the position information is needed at a fixed rate of 10msec. PMAC will always have the data ready for the host in 10msec increments but can the host always be ready for the data? The host must read the data, perhaps do some formatting, and then write the data to a file. If the host has sufficient RAM so that disk access is not required, 10msec should be no problem. However, this of course depends on the host. There will always be a limit to the host’s capability and it will entirely depend on the host, the other jobs the host must do, and the method the host program uses to read and store the data. The question to answer here is what rate is needed for the position information and can the host keep up with the rate?
If the host cannot keep up with the data from the PMAC, there is another possible method. This requires using the DPRAM and PMAC’s data gathering capabilities. On the PMAC side, set up data gathering for the addresses (probably motor positions) wanted and set up the PMAC to place the data into DPRAM in a rotary fashion. The size of the gather buffer (storage size) must also be specified. When the gather command is given to PMAC, it will use a section of DPRAM as the gather buffer and start copying data there. When the end of this DPRAM section is reached, the data insertion point will rollover back to the start of the buffer and fill up the area again. This process can continue indefinitely since the previous buffer contents will just be overwritten. In this scenario, it is the host’s responsibility to get the data before the insertion point rolls over and overwrites previous data. Let us continue the example from above where data was needed at a 10msec increment. Also, assume that the specified data gather buffer size is sufficient for 150 pieces of information before rollover happens. Now, since there is storage, the host can grab a large amount of data, e.g. 100 pieces of information, each second instead of one piece each 10msec. This lessens the load on the host considerably. The drawback to this method is that the host program required to read and convert the data and ensure that it is ahead of the insertion point is much more complicated to write than the program that only gets one piece of information. This feature is called Real-Time Data Gathering through the DPRAM and it is described in the December 1992 version of the PMAC User Manual.

The program below is a sample program that remembers motors positions when a keystroke is received. It would not be difficult to modify this program to use a timer instead of a keystroke. In addition, in the December 1992 version of the PMAC Software Reference, there is a simple program called PMACLERN.C which is the program that the following program is based on.

/* PMACLERN.C
Sample program to implement a learn mode in PMAC. This will generate a motion program (PROG 123) which contains a series of moves which are created each time CTRL-B is pressed.
*/
#include <stdio.h>
#include <dos.h>
#include <conio.h>
#include "comlib.h"
#include "lips.h"
#define ABSOLUTE 0
#define RELATIVE 1
// modes for the getAllPosition function
#define GP_POLLED 0 // Use active com port and do not wait for in position of the CS
#define GP_POLLED_INPOS 1 // Use active com port and wait for in position of the CS
#define GP_POLLED_CAPT 2 // Read captured position registers through active com port
#define GP_DPRAM 3 // Read positions from DPRAM
#define GP_DPRAM_INPOS 4 // Wait for in pos and read positions from DPRAM
#define GP_DPRAM_CAPT 5 // Read captured position registers through DPRAM
int pmac_id, posCaptMode = 0;
char buf[255];
typedef struct{
    float scale[8];
    char axis[8]; }CS_INFO;
CS_INFO cs_info;
typedef struct {
    float position[8];
    float oldPosition[8];
    float scaledPosition[8];
    float oldScaledPosition[8]; }
POS_INFO;
int getCapturedPos (int motor, float *position){
    char address[10];
    int error=1;
    switch (motor){
        case 1:
        case 2:
        case 3:
        case 4:
        case 5:
        case 6:
        case 7:
        case 8:
            sprintf(address,"rx%ld",49151L+4*motor);
            getResponse(pmac_id,buf,255,address);
            stripOffControl (buf); *position = atof(buf);
            break;
        default
            error = -1;
    }
    return(error);
}
int getAllPos (POS_INFO *pos_info, int posMode, int timeOut, int coordSys){
    /* pos_info Structure containing position information
       posMode Switch to determine how to get positions
       0 = Use active com port and do not wait for in position of the CS
       1 = Use active com port and wait for in position of the CS
       2 = Read captured position registers through active com port
       3 = Read positions from DPRAM
       4 = Wait for in pos and read positions from DPRAM
       5 = Read captured position registers through DPRAM
       timeOut How much time to wait for in position before you give up
       coordSys Coordinate system number needed for in position confirming
    */
    int gotPositions = 1;
    static int firstTime = 1;
    long myTimeOut, DPRAMTimeOut, ltemp; float 1temp;
    double dtemp;
    flush(pmac_id);
    switch (posMode){
        case 1: // active port wait for in-pos
            myTimeOut = getTimeout(timeOut);
            // while(!kbhit()){ return(0);
                printf("%s", inPositionAll(pmac_id, coordSys));
                //
            }
            while (inPositionAll(pmac_id, coordSys)==0 && myTimeOut>0){
                myTimeOut--;
                myTimeOut--;
                if (myTimeOut<=0){
                    gotPositions = -1;
                }
                //
        }
break;
}
flush(pmac_id);
controlResponse(pmac_id,buf,255,16);
stripOffControl (buf);
if(strlen(buf) >= 10) // ask for position of motors
for (i=0;i<8;i++)
    // save last positions
    pos_info->oldPosition[i] = pos_info->position[i];
    pos_info->oldScaledPosition[i] = pos_info->scaledPosition[i];
}
sscanf (buf,"%f%f%f%f%f%f%f%f",&pos_info->position[0],&pos_info->position[1],&pos_info->position[2],&pos_info->position[3],&pos_info->position[4],&pos_info->position[5],&pos_info->position[6],&pos_info->position[7]);
for (i=0;i<8;i++)
    // make scaled positions
    if (cs_info.scale[i]!= 0 && cs_info.axis[i]!= 0)
        pos_info->scaledPosition[i] = pos_info->position[i]/cs_info.scale[i];
else
    gotPositions = 0;
break;

case 2: // active port read captured position registers
flush(pmac_id);
for (i=0;i<8;i++)
    // save last positions
    pos_info->oldPosition[i] = pos_info->position[i];
    pos_info->oldScaledPosition[i] = pos_info->scaledPosition[i];
    getCapturedPos (i+1,&ftemp); // read new position
    pos_info->position[i] = ftemp;
    if (cs_info.scale[i]!= 0 && cs_info.axis[i]!= 0)
        pos_info->scaledPosition[i] = ftemp/cs_info.scale[i];
    else
        pos_info->scaledPosition[i] = 0;
break;

case 3: // DPRAM polled
flush(pmac_id);
for (i=0;i<8;i++)
    // save last positions
    pos_info->oldPosition[i] = pos_info->position[i];
    pos_info->oldScaledPosition[i] = pos_info->scaledPosition[i];
}
DPRAMTimeOut = getTO(pmac_id);
DPsetHostBusyBit (pmac_id, TRUE);
while (DPtrGetPmacBusyBit (pmac_id) && i<DPRAMTimeOut)
i++;
if (i>=DPRAMTimeOut)
    return (-2);
for (i=0;i<8;i++)
    // get the new positions
    DPgetActualPos (pmac_id, i+1, 1.0, &dtemp); // read new position
    pos_info->position[i] = (float) dtemp;
}
DPsetHostBusyBit (pmac_id, FALSE);
for (i=0;i<8;i++)
    // scale the new positions
    if (cs_info.scale[i]!= 0 && cs_info.axis[i]!= 0)
        pos_info->scaledPosition[i] = pos_info->position[i]/cs_info.scale[i];
>position[i]/cs_info.scale[i];
  else
    pos_info->scaledPosition[i] = 0;
}
break;

case 4: // DPRAM wait for in-pos
  myTimeOut = getTimeout(timeOut);
  DPRAMTimeOut = getTO(pmac_id);
  DPsetBackground (pmac_id, FALSE);
  while (DPgetBackground (pmac_id) == FALSE && DPRAMTimeOut>0) {
    DPRAMTimeOut--;
  }
  if (i>=DPRAMTimeOut)
    return (-2);

  while (!kbhhit()){
    i=DPsysInposition(pmac_id, coordSys);
    printf("DPsysInposition is %d\r\n",i);
    DPsetBackground (pmac_id, FALSE);
    DPRAMTimeOut=1000;
    while (!DPgetBackground (pmac_id) && DPRAMTimeOut>0) {
      DPRAMTimeOut--;
    }
  }

  /*
   while (DPsysInposition(pmac_id, coordSys)==0 && myTimeOut>0) { // wait for
   in position loop
   //
   myTimeOut--;        
   myTimeOut--;        
   DPsetBackground (pmac_id, FALSE);
   DPRAMTimeOut=getTO(pmac_id);
   while (!DPgetBackground (pmac_id) && DPRAMTimeOut>0) {
     DPRAMTimeOut--;    
   }
   }
  */

  if (myTimeOut<=0) {
    gotPositions = -1;
    break;
  }
}

flush(pmac_id);
for (i=0;i<8;i++) {
  // save last positions
  pos_info->oldPosition[i] = pos_info->position[i];
  pos_info->oldScaledPosition[i] = pos_info->scaledPosition[i];
}

DPRAMTimeOut = getTO(pmac_id);
DPsetHostBusyBit (pmac_id, TRUE);
while (DPgetPmacBusyBit (pmac_id) && i<DPRAMTimeOut) i++;
if (i>=DPRAMTimeOut)
  return (-2);
for (i=0;i<8;i++) {
  // get the new positions
  DPgetActualPos (pmac_id, i+1, 1.0, &dtemp); // read new position
  pos_info->position[i] = (float) dtemp;
}
DPsetHostBusyBit (pmac_id, FALSE);
for (i=0;i<8;i++) {
  // scale the new positions
  if (cs_info.scale[i]!= 0 && cs_info.axis[i]!= 0)
Using PMAC in a Learn Mode

```c
pos_info->scaledPosition[i] = pos_info->position[i] / cs_info.scale[i];
else
    pos_info->scaledPosition[i] = 0;
}
break;

case 5: // captured position in DPRAM background variable buffer
    flush(pmac_id);
    DPsetVBackground(pmac_id, FALSE); // tell PMAC to update the buffer
    for (i=0;i<8;i++){
        // save last positions
        pos_info->oldPosition[i] = pos_info->position[i];
        pos_info->oldScaledPosition[i] = pos_info->scaledPosition[i];
    }
    DPRAMTimeOut = getTO(pmac_id);
    while (!DPgetVBackground(pmac_id) && i<DPRAMTimeOut)
        i++;
    if (i>=DPRAMTimeOut)
        return (-2);
    for (i=0;i<8;i++){
        // get the new positions
        ltemp = DPgetVBGdata(pmac_id, i+1); // read new position
        pos_info->position[i] = (float) ltemp;
    }
    DPsetVBackground(pmac_id, FALSE); // tell PMAC to update the buffer
    for (i=0;i<8;i++){
        // update position array
        if (cs_info.scale[i]!= 0 && cs_info.axis[i]!= 0)
            pos_info->scaledPosition[i] = pos_info->position[i] / cs_info.scale[i];
        else
            pos_info->scaledPosition[i] = 0;
    }
    break;

default: // active port polled
    flush(pmac_id);
    controlResponse(pmac_id,buf,255,16);
    stripOffControl(buf);
    if(strlen(buf) >= 10){ // ask for position of motors
        for (i=0;i<8;i++){
            // save last positions
            pos_info->oldPosition[i] = pos_info->position[i];
            pos_info->oldScaledPosition[i] = pos_info->scaledPosition[i];
        }
        sscanf(buf, "%f%f%f%f%f%f%f%f", &pos_info->position[0],&pos_info->position[1],&pos_info->position[2],&pos_info->position[3],&pos_info->position[4],&pos_info->position[5],&pos_info->position[6],&pos_info->position[7]);
        for (i=0;i<8;i++){
            // make scaled positions
            if (cs_info.scale[i]!= 0 && cs_info.axis[i]!= 0)
                pos_info->scaledPosition[i] = pos_info->position[i] / cs_info.scale[i];
        }
    } else
        gotPositions = 0;
    break;
```
if (firstTime){ // make sure last position != current position
    firstTime = 0;
    for (i=0;i<8;i++) // save last positions
    {
        pos_info->oldPosition[i] = pos_info->position[i]-100;
        pos_info->oldScaledPosition[i] = 0;
    }
}
return (gotPositions);

int addProgLine (int nMoveMode, char *scaling, char *destination){
/* moveMode is either incremental or absolute
   format is either long which adds every axes position
   or short which only adds changed axes positions
   scaling is either scaled which uses the CS scale factor
   or unscaled which does not use the CS scale factor
   destination is the PMAC program "prog n" or file that the program text gets appended to
   Return codes 1 = No errors
   0 to -2 = errors from getAllPositions
   -3 = Invalid mode
   -4 = No new positions
   -5 = Invalid output destination
*/
    int error=1, i, length;
    char program_line[255], moveMode[4],command[255];
    static char oldMoveMode[4] = "x";
    static POS_INFO pos_info;
    FILE *out_file;
    error = getAllPos(&pos_info,posCaptMode,10,1);
    if (error>0){
        error = 1;
        program_line[0]=NULL; // reset the program line
        switch (nMoveMode){
            case 0: // absolute mode
                sprintf(moveMode,"abs");
                for (i=0;i<8;i++) // only add if defined and changed
                {
                    if (*scaling=='s' || *scaling=='S')
                        sprintf
                    else
                        sprintf
                        (buf,"%c%f",cs_info.axis[i],pos_info.scaledPosition[i] - pos_info.oldScaledPosition[i]);
                    else
                        sprintf
                        (buf,"%c%f",cs_info.axis[i],pos_info.position[i] - pos_info.oldPosition[i]);
                    strcat(program_line,buf);
                }
                break;
            case 1. // incremental mode
                sprintf(moveMode,"inc");
                for(i=0;i<8;i++) // only add if defined
                {
                    if(cs_info.axis[i]!=NULL && pos_info.position[i] !=
                        pos_info.oldPosition[i])
                        if (*scaling=='s' || *scaling=='S')
                            sprintf
                        else
                            sprintf
                            (buf,"%c%f",cs_info.axis[i],pos_info.scaledPosition[i] - pos_info.oldScaledPosition[i]);
                    else
                        sprintf
                        (buf,"%c%f",cs_info.axis[i],pos_info.position[i] - pos_info.oldPosition[i]);
                    strcat(program_line,buf);
                }
        }
    return (gotPositions);
}
if(*scaling=='s'||*scaling=='S'  
    sprintf(buf, "%c%f", cs_info.axis[i], pos_info.scaledPosition[i] - pos_info.oldScaledPosition[i]);  
else  
    sprintf(buf, "%c%f", cs_info.axis[i], pos_info.position[i] - pos_info.oldPosition[i]);  
strcat(program_line, buf);  

break; *  
default: // invalid mode  
    error = -3;  
}  
if (program_line[0]!=NULL){  
    length = strlen(destination);  
    for (i=0; i<length; i++)  
       destination[i] = toupper(destination[i]);  
}  
if (strncmp(destination, "PROG", 4)==0 && error==1){  
    printf("%s\n", program_line);  
    sprintf(command,"OPEN %s", destination);  
    getResponse(pmac_id, buf, 255, command);  
    if (strcmp(oldMoveMode, moveMode)!=0){ // moveMode has changed  
        strcpy(oldMoveMode, moveMode);  
        getResponse(pmac_id, buf, 255, moveMode);  
    }  
    getResponse(pmac_id, buf, 255, program_line);  
    getResponse(pmac_id, buf, 255, "CLOSE");  
}  
else if ((out_file = fopen(destination, "r+t"))!=NULL){ // destination is a file in the PC  
    printf("%s\n", program_line);  
    fseek(out_file, 0, SEEK_END);  
    fseek(out_file, ftell(out_file)-7, SEEK_SET);  
    if (strcmp(oldMoveMode, moveMode)!=0){ // moveMode has changed  
        strcpy(oldMoveMode, moveMode);  
        fprintf(out_file,"%s\n", moveMode);  
    }  
    fprintf(out_file,"%s\n", program_line);  
    fprintf(out_file,"CLOSE\n");  
    fclose(out_file);  
    error = -5; // invalid output destination  
}  
else  
    error = -4; // no new positions  
}  
// else  
//   error = -3;  
return(error);  
}  
int getAxes (int coordinateSystem){  
/*int nNameOfVariable;  
long ILongVariable;  
char szBuffer[10];  

Application Note
Using PMAC in a Learn Mode
float fName;
/*
    int i,firstChar=0,foundOne=0,cs;
    char *cp;
    for (i=0;i<8;i++){
        getAxisDefinition(pmac_id, i+1, &cs, buf);
        if (cs==coordinateSystem){
            cp = buf;
            firstChar = 0;
            while ((*cp>90 || *cp<65) && *cp!=NULL){
                if (*cp<32) // check for leading control characters
                    firstChar++;
                cp++;
            }
            if (*cp==NULL){
                printf("error reading axis %d definition
",i+1);
                cs_info.axis[i] = NULL;
            }
            else{
                foundOne=1;
                cs_info.axis[i] = *cp;
                *cp = NULL;
                cp = &buf[firstChar];
                if (*cp==NULL)
                    cs_info.scale[i] = 1.0;
                else
                    cs_info.scale[i] = (float) atof(&buf[firstChar]);
                printf("axis %d in C.S %d has a scale factor of
%f\n",i+1,coordinateSystem,cs_info.scale[i]);
            }
        }
    }
    return (foundOne);
}
*/
int getAxes (int coordinateSystem){
    // int nNameOfVariable;
    // long lLongVariable;
    // char szBuffer[10];
    // float fName;
    int i,firstChar=0,foundOne=0;
    char *cp,command[255];
    for (i=0;i<8;i++){
        sprintf(command,"&%d#%d->",coordinateSystem,i+1);
        getResponse(pmac_id,buf,255,command); // ask for definition of motor 1
        cp = buf;
        firstChar = 0;
        while ((*cp>90 || *cp<65) && *cp!=NULL){
            if (*cp<32) // check for leading control characters
                firstChar++;
            cp++;
        }
    }
    return (foundOne);
}
```c
void parseError (int error)
{
    switch (error) {
    case 0:
        printf("Timed out reading positions\n");
        break;
    case -1:
        printf("Timed out on in-position\n");
        break;
    case -2:
        printf("Timed out reading DPRAM\n");
        break;
    case -3:
        printf("Invalid program mode\n");
        break;
    case -4:
        printf("No new motor positions\n");
        break;
    case -5:
        printf("Invalid output destination\n");
        break;
    case -6:
        printf("Unable to list program file\n");
        break;
    default:
        printf("Undocumented error #\n",error);
    }
}

void main () {
    int done = 0, error, i;
    char ic,buf[255],tbuf[255],destination[30];
    long int VAddrArray [8];
    FILE *out_file;
    sprintf(destination, "PROG 123"); // default output
    buf[0] = NULL;
    // ...
```
tbuf[0] = NULL;
printf ("Initializing communications with PMAC.\n");
printf (" Opening ComLib.\n");
pmac_id = CLinit(NULL,"C:\windows\PMACCOM.INI");
getResponse(pmac_id,buf,255,"i3=2");
printf (" Opening LIPS.\n");
openLIPS (pmac_id);
if(pmac_id,0){
printf(" PMAC not found, error = %d",pmac_id);
exit(0);
}
printf ("Initializing DPRAM real time buffer.\n");
if(DPrealTime (pmac_id, 30, TRUE)!=0){
printf(" DPRAM real time buffer initialized.\n");
DPsetMotors (pmac_id, 8);
}
else
printf(" DPRAM real time buffer not found.\n");
if(DPbackground (pmac_id, TRUE)!=0)
printf(" DPRAM fixed data background buffer initialized.\n");
else
printf(" DPRAM fixed data background buffer not found.\n");
printf ("Initializing DPRAM variable data background buffer.\n");
for (i=0;i<8;i++){
    VAddrArray[i] = 0x0002c003L + i*4; // motor "i+1" captured position registers
}
if(DPvarBufUpdt (pmac_id, 8, VAddrArray)!=0)
printf(" DPRAM variable data background buffer initialized.\n");
else
printf(" DPRAM variable data background buffer not found.\n");
if (pmac_id>0) {
printf ("Now acting as a terminal to PMAC.\n");
printf ("Press <ESC> to abort.\n");
printf ("Now creating program 123.\n");
getResponse(pmac_id,buf,255,"OPEN PROG 123 CLEAR CLOSE");
setEndChar (pmac_id, 10, 10, 13);
getAxes (1);
while (!done) {
    while (!kbhit()) { // no key pressed, so check card for data
        if (getLine(pmac_id,buf,255)){
            stripOffControl (buf);
            printf ("%s\r\n",buf);
        }
    }
    switch (ic = getch ()){
        case 2: // CTRL-B pressed, so add abs moves to program
            error = addProgLine (ABSOLUTE, "scaled", destination);
            break;
        // Other cases...
    }
    if (getLine(pmac_id,buf,255)){
        stripOffControl (buf);
        printf ("%s\r\n",buf);
    }
}}
if (error<1)
    parseError (error); flush(pmac_id);
break;
case 4: // CTRL-D pressed
    if (strncmp(destination, "PROG", 4)==0){
        printf ("\nLearned moves in PROG 123:\n\n");
        printf ("OPEN PROG 123 CLEAR\n\n");
        sendLine(pmac_id,"LIST PROG 123");
    }
    else if ((out_file = fopen (destination, "rt"))!=NULL){ // destination is a file in the PC
        printf ("\nLearned moves in file %s/r/n",destination);
        fseek (out_file, 0, SEEK_SET);
        while (fgets (buf,254,out_file)!=NULL){
            printf (buf);
        }
        fclose (out_file);
    }
    else{
        error = -6; // invalid output destination
        parseError (error); } break;
case 11: // CTRL-K pressed, so add inc moves to program
    error = addProgLine (RELATIVE, "scaled", destination);
    if (error<1)
        parseError (error);
    flush(pmac_id);
break;
case 27: // ESC key pressed, so exit
    done = 1;
    break;
case 0: // get 2nd part of special keys (i.e. F-keys)
    switch (ic = getch ()){
        case 59:// Use active com port and do not wait for position of the CS
            printf ("\nF%d was pressed\n",ic-58);
            printf ("Using bus port and not waiting for in position\n");
            posCaptMode = GP_POLLED;
            break;
        case 60: // Use active com port and wait for in position of the CS
            printf ("\nF%d was pressed\n",ic-58);
            printf ("Using bus port and waiting for in position\n");
            posCaptMode = GP_POLLED_INPOS;
            break;
        case 61: // Read captured position registers through active com port
            printf ("\nF%d was pressed\n",ic-58);
            break;
        case 62: // Read positions from DPRAM
            printf ("\nF%d was pressed\n",ic-58);
            printf ("Using DPRAM and not waiting for in position\n");
            posCaptMode = GP_DPRAM;
            break;
        case 63: // Wait for in pos and read positions from DPRAM
            printf ("\nF%d was pressed\n",ic-58);
            printf ("Using DPRAM and waiting for in positions\n");
            posCaptMode = GP_DPRAM_INPOS;
            break;
        case 64: // Read captured position registers through DPRAM
            break;
    }
}
printf ("\r\nF%d was pressed\r\n",ic-58);
printf ("Using DPRAM and reading captured position\r\n");
posCaptMode = GP_DPRAM_CAPT;
break;
case -123: // Direct output to PMAC
    printf ("\r\nF%d was pressed\r\n",ic+134);
    printf ("Output directed to PMAC program123\r\n");
sprintf(destination, "PROG 123"); // default output
    out_file = fopen (destination, "wt"); // destination is a file in the PC
    fprintf(out_file,"OPEN PROG 123 CLEAR\r\n");
    fclose (out_file);
    break;
case -122: // Direct output to a file
    printf ("\r\nF%d was pressed\r\n",ic+134);
    printf ("Output directed to file testProg.pmc\r\n");
sprintf(destination, "y:\brad\learn\testProg.pmc"); // default output
    out_file = fopen (destination, "wt"); // destination is a file in the PC
    fprintf(out_file,"OPEN PROG 123 CLEAR\n");
    fprintf(out_file,"CLOSE\n");
    fclose (out_file);
    break;
default: // Use active com port and do not wait for in position of the CS
    printf ("\r\nF%d was pressed\r\n",ic-58);
    printf ("Using bus port and not waiting for in position\r\n");
posCaptMode = GP_POLLED_CAPT;
    break;
}
break;
default:
    // sendChar (pmac_id,ic); // send character to PMAC
    if (ic==13){
        sendLine (pmac_id, tbuf);
        tbuf[0] = NULL;
putch(10);
    }
    else
    strcat (tbuf,&ic);
putch (ic);
    break;
}
}
else
    printf ("Can't talk to PMAC!\r\n");
CLclose(pmac_id);
}