PCStream

PCStream is an imaginary I/O board for the SWTPCemu, SWTPCemuApp and memulator 6800/6809 emulators. This board is used to transfer data between the host PC and the emulated system via a data stream. The board uses two addresses in the emulated processor's memory space. The first address space is used for checking status and issuing commands. This address (port) is used to set various flags that control the data stream. The second address (port) has two registers, one for data and the other for command type.

In order to be able to write to the data port as both a command type register and a data type register, there is a command that will set the mode. To set the mode for writing to the command/control register of the data port send the command 0x03 to the command/status port. To set for writing data to the data stream the command is 0x04. This only makes sense for using the port to write to streams since all data written to the data port while reading a stream is command/control info. The default after a reset is to write to the command/control register.

Status/Command Port (all values are specified in hexadecimal):

00 Reset the controller.

1. Resets the filename is set flag
2. Convert line feed to carriage return on read from PC is set false
3. Discard line feeds on read from PC is set to false
4. Add line feed after carriage return on write to PC is set to false
5. Data port mode is set to command/control

01 set flag to convert line feed to carriage return on read from PC

02 discard line feeds on read from PC

03 set data port mode when writing to writing to controller control registers for specifying the filename to write to. This command must be immediately followed by writing to the controller's data port the null terminated filename for opening a file on the PC or directory name for listing or changing directory on the PC.

04 set up the controller for actually writing to the actual data stream (file opened for write) this command must be sent before actual writing to PC is allowed otherwise writing will continue to go to command/control port. Once this command is sent, all writing to the data port will go through the stream until the controller is reset.

05 Set flag to expand tabs to spaces when writing to the PC

06 Set flag to add line feeds to carriage return when writing to PC

After the file/directory name has been sent to the controller and the file/dir name has been terminated, the next byte sent to the controller's data/control must be the command that specifies what action the controller should take.

01	read binary
02	read text
03	write binary
04	write text
05	append binary
06	append text
07	delete
08	close
10	return directory listing
11	change working directory
12	reset current working directory to the applications current working directory

If the 0x10 command (return working directory) is sent without a directory name sent (just the null terminator after the 0x03 command), the currently set current directory is assumed. The directory name can be either and absolute path with drive letter starting the name, a UNC path that starts with // or a path relative to the current working directory. Whenever specifying a directory or filename for the PC side, the same rule applies. Capitalization does not matter and for path separation either '/' or '\' can be used unless specifying a UNC path. In that case '\' must be used.

PCSTREAM.C – sample c code of how to use PCStream I/O Device

```
#include <stdio.h>

char *PCStream = 0xE060;
char currentWorkingDirectory [512];
char flexFilename[16];
char pcFilename[512];

int ListFile (filename)
char* filename;
{
  char c;

  *PCStream = 0x00;
  *PCStream = 0x03;
  while (*filename != 0x00)
  {
    *(PCStream+1) = *filename;
    filename++;
  }
  *(PCStream+1) = 0x00;
  *(PCStream+1) = 0x02;  /* read file */

  while (*PCStream == 0x00)
  {
    c = *(PCStream+1);
    if (*PCStream == 0x00)
      putchar (c);
  }

  return (0);
}

int GetDirectory ()
{
  char c;

  *PCStream = 0x00;
  *PCStream = 0x03;
  *(PCStream+1) = 0x00;
  *(PCStream+1) = 0x10;
  while (*PCStream == 0x00)
  {
    c = *(PCStream+1);
    if (*PCStream == 0x00)
      putchar (c);
  }
  return (0);
}

int GetCurrentWorkingDir ()
{
  char c;
  int i;

  *PCStream = 0x00;
  *PCStream = 0x03;
  *(PCStream+1) = 0x00;
  *(PCStream+1) = 0x09;

  for (i = 0; i < 511, *PCStream != 0x00; i++)
  {
    c = *(PCStream+1);
    currentWorkingDirectory[i] = c;
    currentWorkingDirectory[i + 1] = 0x00;
  }
  return (i);
```

```
}

int ChangeDirectory (dir)
char *dir;
{

  *PCStream = 0x00;
  *PCStream = 0x03;
  while (*dir != 0x00)
  {
    *(PCStream+1) = *dir;
    dir++;
  }
  *(PCStream+1) = 0x00;
  *(PCStream+1) = 0x11;

  return (*PCStream);
}

int ResetCurrentDirectory ()
{
  *PCStream = 0x00;
  *PCStream = 0x03;
  *(PCStream+1) = 0x00;
  *(PCStream+1) = 0x12;

  return (0);
}

int WriteTextFileToFLEX (srcFilename, tgtFilename)
char *srcFilename;
char *tgtFilename;
{
  FILE *fp = NULL;
  char c;
  int status;

  *PCStream = 0x00;   /* reset port */
  *PCStream = 0x03;   /* get controller ready to accept a filename */

  /* send target file name to controller */

  while (*srcFilename != 0x00)
  {
    *(PCStream+1) = *srcFilename;
    srcFilename++;
  }
  *(PCStream+1) = 0x00;   /* terminate the send of the filename */

  *PCStream = 0x01;     /* set flag to convert line feed to carriage return*/
  *PCStream = 0x02;     /* set flag to throw awy line feeds */
  *(PCStream+1) = 0x02;   /* read file */

  fp = fopen(tgtFilename, "w");
  if (fp != NULL)
  {
    while (*PCStream == 0x00)
    {
      c = *(PCStream+1);
      if (*PCStream == 0x00)
        putc (c, fp);
    }
    status = *PCStream;
  }
  else
    status = 0xff;

  *PCStream = 0x00;   /* reset port */
  return (status);
}

int WriteTextFileToPC (srcFilename, tgtFilename)
char *srcFilename;
char *tgtFilename;
{
  int c;
  FILE *srcFp;
  int status;

  *PCStream = 0x00;   /* reset the port for new activity */
  *PCStream = 0x03;   /* get controller ready to accept a filename */

  /* send target file name to controller */

  while (*tgtFilename != 0x00)
  {
    *(PCStream+1) = *tgtFilename;
    tgtFilename++;
  }
  *(PCStream+1) = 0x00;   /* terminate the send of the filename */
  *(PCStream+1) = 0x04;   /* command to write to stream */
  *PCStream = 0x04;   /* allow writing to the stream */

  /* required only if opening file in binary mode within FLEX */
  /* and you want to do space expansion - otherwise not needed */

  *PCStream = 0x85;   /* turn on space expansion */

  /* open the source file */

  srcFp = fopen(srcFilename, "r");
  if (srcFp != NULL)
  {
    while ((c = getc(srcFp)) != -1)
    {
      *(PCStream+1) = c & 0x00ff;
    }
    status = *PCStream;
    *PCStream = 0x00;   /* re-init port and close file */
  }
  else
    status = 0xff;

  return (status);
}

gotorc (r, c)
int r, c;
{
  putchar(0x1b);
  putchar('=');
  putchar(r + 0x20);
  putchar(c + 0x20);
}

int ShowMenu ()
{
  int sizeOfCWD;
  int j;

  sizeOfCWD = GetCurrentWorkingDir();

  printf("%c             PCStream File Transfer Program", 0x1a);
  gotorc ( 1, (80 - sizeOfCWD) / 2);
  printf(currentWorkingDirectory);
  gotorc ( 3, 10); printf ("1.  List File");
  gotorc ( 4, 10); printf ("2.  List Directory");
  gotorc ( 5, 10); printf ("3.  Copy File To PC");
  gotorc ( 6, 10); printf ("4.  Copy File From PC");
  gotorc ( 7, 10); printf ("5.  Change Working Directory");
  gotorc ( 8, 10); printf ("6.  Show Working Directory");
  gotorc ( 9, 10); printf ("7.  Reset Working Directory");
  gotorc (10, 10); printf ("X.  Exit");
  gotorc (18, 10); printf ("Enter command");
}

int AnyKeyToContinue ()
{
  gotorc (23, 0); printf("press any key to continue");
  return (getchar());
}

int Error (errorMessage) char *errorMessage;
{
  gotorc (21, 0); printf(errorMessage);
  AnyKeyToContinue ();
}

int main (argc, argv)
int argc;
char **argv;
{
  int command;
  char getsBuffer[1024];
  int bytesInputToGetsBuffer;
  int commandStatus;

  int status;

  do
  {
    ShowMenu();
    command = getchar();

    switch (command)
    {
      case '1':
        gotorc (20, 0); printf("Enter PC filename: ");
        bytesInputToGetsBuffer = gets (getsBuffer);
        if (getsBuffer[0] != 0x00)
        {
          commandStatus = ListFile(getsBuffer);
          if (commandStatus != 0x00)
          {
            Error ("Error listing file");
          }
        }
        break;
      case '2':
        putchar(0x1a);
        GetDirectory ();
        AnyKeyToContinue();
        break;
      case '3':
        gotorc (20, 0); printf("Enter FLEX filename: "); gets (flexFilename);
        gotorc (21, 0); printf("Enter PC   filename: "); gets (pcFilename);
        if (flexFilename[0] != 0x00)
        {
          if (pcFilename[0] == 0x00)
          {
            /* no target filename specified - use source file name as target file name*/
            strcpy (pcFilename, flexFilename);
          }
          WriteTextFileToPC(flexFilename, pcFilename);
        }
        else
```

```
          Error("You must provide a FLEX filename to copy");
        break;
      case '4':
        gotorc (20, 0); printf("Enter PC   filename: "); gets (pcFilename);
        gotorc (21, 0); printf("Enter FLEX filename: "); gets (flexFilename);
        if (pcFilename[0] != 0x00)
        {
          if (flexFilename[0] != 0x00)
          {
            status = WriteTextFileToFLEX(pcFilename, flexFilename);
            if (status != 0x04)   /* make sure return status = EOF */
            {
              /* Error("Bad status returned from WriteTextFileToFLEX"); */
              gotorc (21, 0); printf("Bad status returned from WriteTextFileToFLEX:");
              printf (" %02X", status);
              AnyKeyToContinue ();
            }
          }
          else
            Error("You must provide a FLEX filename to copy");
        }
        else
          Error("You must provide a PC filename to copy");
        break;
      case '5':
        gotorc (20, 0); printf("Enter new working directory: ");
        bytesInputToGetsBuffer = gets (getsBuffer);
        if (getsBuffer[0] != 0x00)
        {
          commandStatus = ChangeDirectory (getsBuffer);
          if (commandStatus != 0x00)
          {
            Error ("Error setting new working directory");
          }
        }
        break;
      case '6':
        GetCurrentWorkingDir ();
        gotorc (22, 0); printf(currentWorkingDirectory);
        AnyKeyToContinue ();
        break;
      case '7':
        ResetCurrentDirectory ();
        break;

      default:
        break;
    }

  } while ((command != 'X') && (command != 'x'));
}
```