MAY THE FORTH BE
WITH OPTO2 PAC
AND LINUX

INDUSTRIAL CONTROLLERS: PAC,PLC & LINUX

Linux support for industrial hardware like PAC or PLC, in the case of the big and well known brands is basically non-existent. Some hardware is based on "exotic" architectures, however the vast majority have in their guts an x86 or ARM based system (Linux friendly architectures!), but the development and management tools remain developed for Windows operating system!

There are so many reasons (commercial, technical, philosophic) that perpetuates same situation. Searching on the web, we can find some PLC/PAC related applications developed for Linux, but practically all of them are communication drivers (for the open and well documented protocols!) , developed by enthusiasts

OPTO22, FORTH

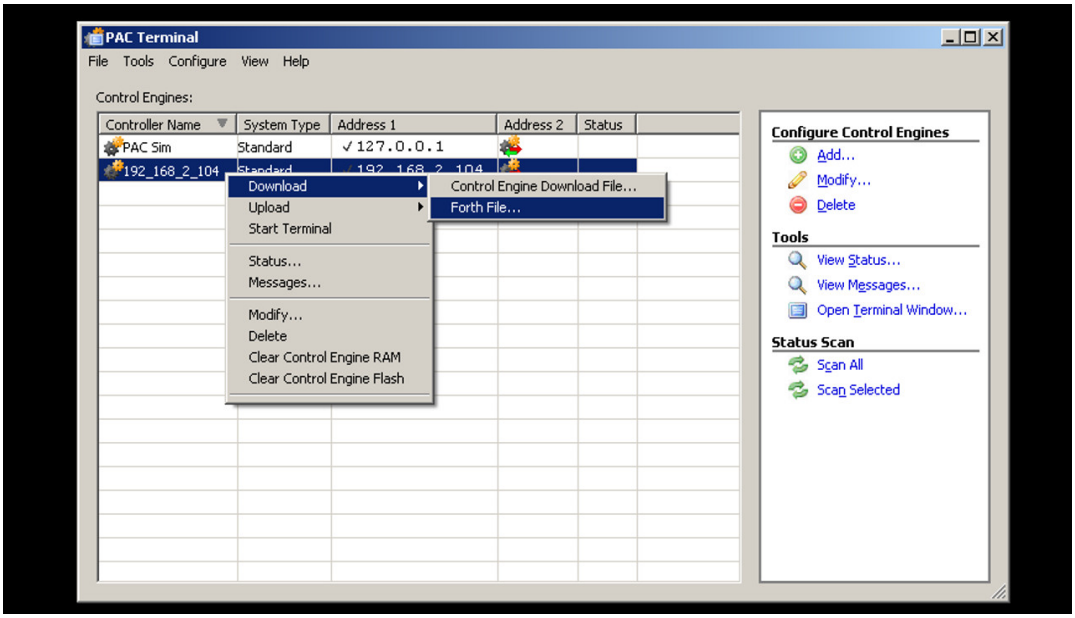
Some time ago, developing a little implementation of RFB protocol in an PAC, in a Wireshark session deliberate unintentionally left open, when uploading a new program, some clear text strings were visible, probably control commands or parts of the program being downloaded!.

Digging a bit more in the application for program loading, an interesting option was discovered: "Download FORTH File". Searching in the web "Opto22" and "FORTH" throws some papyri scanned manuals: How to program Opto22 LC2/LC4 (very outdated hardware!) controllers in FORTH!

With that document, the next natural thing was to open all the generated files with a HEX editor and search some FORTH code inside. The results of the analysis were:

- Manufacturer's application generates many types of files (config, initializations, program code)
- All files were encoded in clear text.
- The programming language used wasn't strictly FORTH

Finding that not a single standard programming language was used to program a PAC is a little pitfall, however, finding that all files were encoded in clear text are good news for programming PACs from non factory supported operating systems like Linux, Mac, etc.



DOWNLOADING PROGRAMS

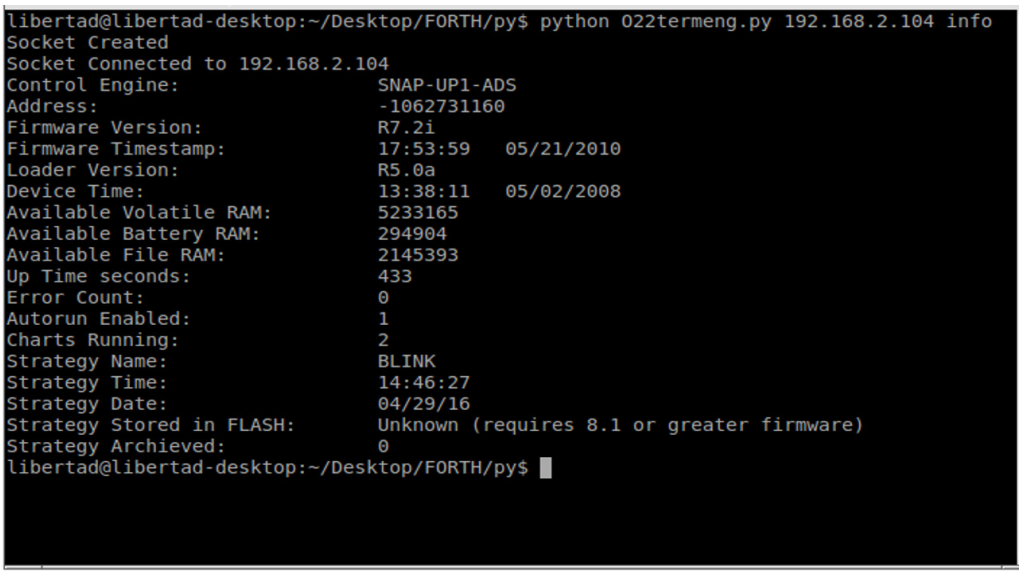
The first step for an approaching to an alternative programming tool, is to know how the original application download an already created program. A simple blink application was used and the way the application talks with the PAC was analyzed using Wireshark.

After downloading many different programs, the event sequence was discovered:

- A F (Handshake?) commands sent
- AcquireLC (Session Lock command?) sent (probably to avoid other applications accessing the PAC while downloading)
- .crn1 file sent line by line
- .crn2 file sent line by line
- .ccd task files sent line by line
- .crn3 file sent line by line
- DATESTAMP sent (control strategy file last modification date)
- TIMESTAMP sent (control strategy file last modification time)
- CRCSTAMP sent (seems like 00112233445566778899AABBCCDDEEFF value means: "ignore" or "don't use" CRC)
- MAKECHECK, CLEAR.BREAKS, [ABORT (Finishing commands?) sent
- ReleaseLC (Session unlock command?) sent

Also, there are a couple of additional commands like get PAC info (available memory, errors, etc), stop program, start program, erase program, etc. that were investigated and implemented!

A python app named O22termeng was developed, is capable of downloading programs to the PAC and send the aforementioned commands



GENERAL STRUCTURE OF A PROGRAM

Now, having a very basic PAC management tool, the control strategy files were studied in detail, and the info gathered is the following:

- All files with extension .crn1, .crn2, .crn3, should have the same name as the "main" program
- .crn1 file contains the line : FILENAME ." PNAME " ; where PNAME is the "main" program name. The rest of the lines of this file never changed when different programs were created
- crn2 files contain the tasks or "charts" names than composes a program, the variables to be used and I/O pin assignation
- .crn3 files contains initializations for: tasks, variables and I/O hardware
- There should be a .ccd file for every task that composes a program, the name should coincide with the information contained in the .crn2 file

A sample control strategy "BLINK" was created. It is composed of two tasks: One toggles a digital output and increments a counter every second. The second one look if the counter is exactly divisible by 5 (modulo 5) and turns on a digital output if the condition is valid, if not turns it off

BLINK.crn1

In this file there is a line where lies the name of the program " BLINK ". The rest of the lines look like some kind of memory space assignation for variables and I/O. No changes were noticed in those lines with different downloaded and analyzed programs.

```
_END NEW $$$.RUN
: FILENAME ." BLINK " ;
1 0 $VAR *_HSV_SEMA
1024 0 $VAR *_HSV_TEMP
200 0 $VAR *_HSV_INIT_IO
0 IVAR ^_HNV_INIT_IO
```

BLINK.crn2

In this file three things were found:

- Tasks that compose the program: Powerup, slowblink, _INI_IO. Powerup is the default task, can't be renamed nor deleted. slowblink was created in the manufacturer program, and _INIT_IO is probably is an internal task that initializes hardware automatically
- Variables used in the program: countervar and modvar
- I/O hardware pin alias: In this particular case digital pins 0 and 3 were assigned alias: DIGITAL_OUT_1 and DIGITAL_OUT_4

```
0 TASK &_INIT_IO
0 TASK &Powerup
0 TASK &slowblink
0 IVAR ^countervar
0 IVAR ^modvar


$0000000000000000.. 32769 1.000000 0.010000 0.000000 2001
$7F000001 0 MBOARD %SELF_RACK
SPOINT 0.0 OUTPUT 0 %SELF_RACK
0 POINT ~DIGITAL_OUT_1
SPOINT 0.0 OUTPUT 3 %SELF_RACK
0 POINT ~DIGITAL_OUT_4
```

BLINK.crn3

This file is a little bit longer than the previous. Seems like in the initial part, tasks, and all types of variables are grouped in arrays and near the end initialization of countervar and modvar to 0 are visible. Notice also that the digital pins are by default inputs "0", and the ones that will be used as outputs must be written to "1" ($..0009 pins 0 and 3)

```
: W_INIT_IO
CONFIG_PORTS
$0000000000000009.. %SELF_RACK ENABLES!
" %SELF_RACK (1/1)" *_HSV_INIT_IO $MOVE 0 ^_HNV_INIT_IO @!
%SELF_RACK ENABLE
" Initializing variables" *_HSV_INIT_IO $MOVE
0 ^countervar @!
0 ^modvar @!
" " *_HSV_INIT_IO $MOVE
```

Task files (Powerup.ccd, slowblink.ccd)

In these files lies the program code, for each task programmed. There are two things here: code blocks, and code block list near the end.

```
T: T0
DUMMY
0_0
0_1
0_3
0_5
0_7
0_10
T;
&Powerup ' T0 SETTASK
```

Each block is labeled a_b, where a is task number and b is a "block id" inside that particular task. Block numbers coincide with the block numbers shown in the manufacturer's application and probably are used for the block-by-block debugging mode.

The following is an example of a code block:

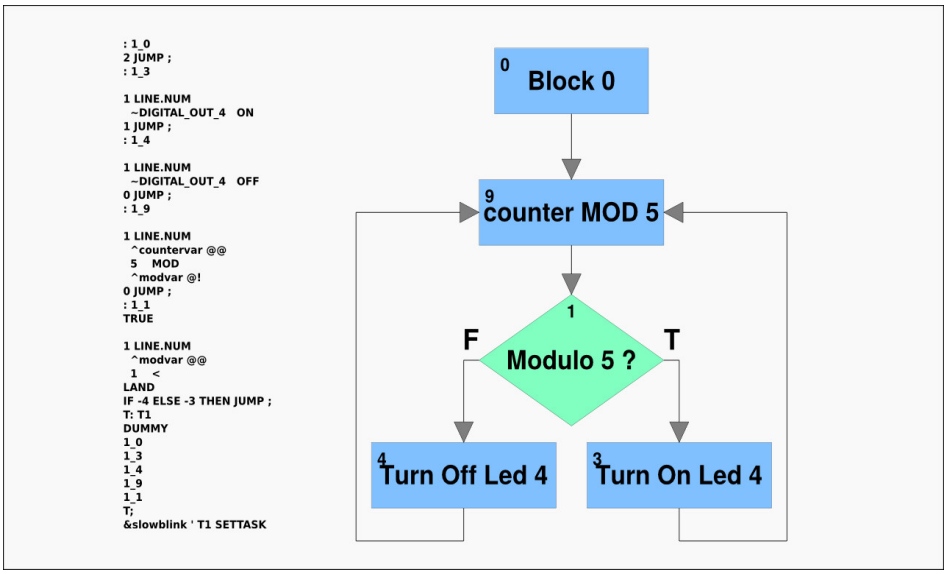
```
: 1_1
TRUE

1 LINE.NUM
^modvar @@
1 <
LAND
IF -4 ELSE -3 THEN JUMP ;
```

Each label is preceded by a colon (:), next LINE.NUM instruction preceded by a line number. This line number coincides with line numbers shown in the manufacturer's development app and are probably used for line-by-line debugging mode

Some portions of the code, closely resemble FORTH specially the IF statement where the condition was BEFORE the word "IF" !

A block finishes with a JUMP instruction followed by a semicolon (;). JUMP instructions are relative, not absolute.



CONCLUSIONS

- It is feasible to build an application to program industrial grade hardware (at least in this specific case with Opto22 PAC) using alternative operating systems!
- The manufacturer didn't use a single standard programming language for the control strategy files, making a little bit harder the task of making an open source tool.
- A skeleton file could be modified to make more complex programs
- There are LOTS of unknown things to investigate.

LINKS

Video Downloading sample strategy to the controller, getting status info, and modifying program: **http://youtu.be/WvSQramo6Cw**

O22termeng.py

```
#Absolutelyautomation.com

#!/usr/bin/python

import sys
import socket
import os.path
import time


def cmdstoreflash( soc ):
     message ="A\r"
     try :
          soc.sendall(message)
     except socket.error:
          print 'Send failed'
          sys.exit()

     reply = soc.recv(2)

     if reply == '\x00\x00':
          message ="F\r"
          try :
               soc.sendall(message)
          except socket.error:
               print 'Send failed'
               sys.exit()

          reply = soc.recv(4)
          # Don't know  exactly what to expect as response !!!
          if reply == '\x00\x00\x00\x00':
               print ' Invalid response for F command? '
               sys.exit()

     else:
          print ' Invalid response for A command? '
          sys.exit()

     message ="BurnIt .\r"
     try :
          soc.sendall(message)
     except socket.error:
```

```
          print 'Send failed'
          sys.exit()

     reply = soc.recv(4)

     if reply <> '\x00\x00\x30\x20':
          print 'Invalid response for BurnIt . command'
          sys.exit()
     print 'Strategy stored in FLASH'

     return


def cmdenableauto( soc ):
     message ="A\r"
     try :
          soc.sendall(message)
     except socket.error:
          print 'Send failed'
          sys.exit()

     reply = soc.recv(2)

     if reply == '\x00\x00':
          message ="F\r"
          try :
               soc.sendall(message)
          except socket.error:
               print 'Send failed'
               sys.exit()

          reply = soc.recv(4)
          # Don't know  exactly what to expect as response !!!
          if reply == '\x00\x00\x00\x00':
               print ' Invalid response for F command? '
               sys.exit()

     else:
          print ' Invalid response for A command? '
          sys.exit()

     message ="1 I!AUTORUN\r"
     try :
          soc.sendall(message)
     except socket.error:
```

```
          print 'Send failed'
          sys.exit()

     reply = soc.recv(2)

     if reply <> '\x00\x00':
          print 'Invalid response for 1 I!AUTORUN command'
          sys.exit()
     print 'Autorun Enabled'

     return

def cmddisableauto( soc ):
     message ="A\r"
     try :
          soc.sendall(message)
     except socket.error:
          print 'Send failed'
          sys.exit()

     reply = soc.recv(2)

     if reply == '\x00\x00':
          message ="F\r"
          try :
               soc.sendall(message)
          except socket.error:
               print 'Send failed'
               sys.exit()

          reply = soc.recv(4)
          # Don't know  exactly what to expect as response !!!
          if reply == '\x00\x00\x00\x00':
               print ' Invalid response for F command? '
               sys.exit()

     else:
          print ' Invalid response for A command? '
          sys.exit()

     message ="0 I!AUTORUN\r"
     try :
          soc.sendall(message)
     except socket.error:
          print 'Send failed'
```

```
          sys.exit()

     reply = soc.recv(2)

     if reply <> '\x00\x00':
          print 'Invalid response for 0 I!AUTORUN command'
          sys.exit()
     print 'Autorun Disabled'

     return

def cmdrun( soc ):
     message ="A\r"
     try :
          soc.sendall(message)
     except socket.error:
          print 'Send failed'
          sys.exit()

     reply = soc.recv(2)

     if reply == '\x00\x00':
          message ="F\r"
          try :
               soc.sendall(message)
          except socket.error:
               print 'Send failed'
               sys.exit()

          reply = soc.recv(4)
          # Don't know  exactly what to expect as response !!!
          if reply == '\x00\x00\x00\x00':
               print ' Invalid response for F command? '
               sys.exit()

     else:
          print ' Invalid response for A command? '
          sys.exit()

     message ="_END _RUN\r"
     try :
          soc.sendall(message)
     except socket.error:
          print 'Send failed'
          sys.exit()
```

```
     reply = soc.recv(2)

     if reply <> '\x00\x00':
          print 'Invalid response for _END_RUN command'
          sys.exit()
     print 'Strategy Running'

     return

def cmdstop( soc ):
     message ="A\r"
     try :
          soc.sendall(message)
     except socket.error:
          print 'Send failed'
          sys.exit()

     reply = soc.recv(2)

     if reply == '\x00\x00':
          message ="F\r"
          try :
               soc.sendall(message)
          except socket.error:
               print 'Send failed'
               sys.exit()

          reply = soc.recv(4)
          # Don't know  exactly what to expect as response !!!
          if reply == '\x00\x00\x00\x00':
               print ' Invalid response for F command? '
               sys.exit()

     else:
          print ' Invalid response for A command? '
          sys.exit()

     message ="_END\r"
     try :
          soc.sendall(message)
     except socket.error:
          print 'Send failed'
          sys.exit()
```

```
     reply = soc.recv(2)

     if reply <> '\x00\x00':
          print 'Invalid response for _END command'
          sys.exit()
     print 'Strategy Stopped'

     return


def cmderaseram( soc ):
     message ="A\r"
     try :
          #Set the whole string
          soc.sendall(message)
     except socket.error:
          #Send failed
          print 'Send failed'
          sys.exit()

     #Now receive data
     reply = soc.recv(2)
     if reply == '\x00\x00':
          message ="EMPTY\r"
          try :
               #Set the whole string
               soc.sendall(message)
          except socket.error:
               #Send failed
               print 'Send failed'
               sys.exit()

          #Now receive data
          reply = soc.recv(2)
          if reply == '\x00\x00':
               print 'RAM erased '
          else:
               print 'error erasing RAM '

     else:
          print 'error erasing RAM '
     return;

def cmderaseflash( soc ):
     message ="A\r"
```

```
     try :
          #Set the whole string
          soc.sendall(message)
     except socket.error:
          #Send failed
          print 'Send failed'
          sys.exit()

     #Now receive data
     reply = soc.recv(2)
     if reply == '\x00\x00':
          message ="EraseIt\r"
          try :
               #Set the whole string
               soc.sendall(message)
          except socket.error:
               #Send failed
               print 'Send failed'
               sys.exit()

          #Now receive data
          reply = soc.recv(2)
          if reply == '\x00\x00':
               print 'FLASH erased '
          else:
               print 'error erasing FLASH '

     else:
          print 'error erasing FLASH '
     return;

def cmdinfo( soc ):
     message ="A\r"
     try :
          soc.sendall(message)
     except socket.error:
          print 'Send failed'
          sys.exit()

     reply = soc.recv(2)

     if reply == '\x00\x00':
          message ="F\r"
          try :
               soc.sendall(message)
```

```
        except socket.error:
             print 'Send failed'
             sys.exit()

        reply = soc.recv(4)
        # Don't know  exactly what to expect as response !!!
        if reply == '\x00\x00\x00\x00':
             print ' Invalid response for F command? '
             sys.exit()

   else:
        print ' Invalid response for A command? '
        sys.exit()


   message ="$WhoAmI?\r"
   try :
        soc.sendall(message)
   except socket.error:
        print 'Send failed'
        sys.exit()

   reply = soc.recv(30)
   print 'Control Engine: '+'\t\t'+reply

   message ="MY.ADDRESS .\r"
   try :
        soc.sendall(message)
   except socket.error:
        print 'Send failed'
        sys.exit()

   reply = soc.recv(30)
   print 'Address: '+'\t\t\t'+reply

   message ="Rev\r"
   try :
        soc.sendall(message)
   except socket.error:
        print 'Send failed'
        sys.exit()

   reply = soc.recv(30)
   print 'Firmware Version: '+'\t\t'+reply
```

```
message ="RevTime  .\"    \" RevDate\r"
try :
     soc.sendall(message)
except socket.error:
     print 'Send failed'
     sys.exit()

reply = soc.recv(30)
print 'Firmware Timestamp: '+'\t\t'+reply

message ="LoaderRev\r"
try :
     soc.sendall(message)
except socket.error:
     print 'Send failed'
     sys.exit()

reply = soc.recv(30)
print 'Loader Version: '+'\t\t'+reply

message ="PTIME  .\"    \" PDATE\r"
try :
     soc.sendall(message)
except socket.error:
     print 'Send failed'
     sys.exit()

reply = soc.recv(30)
print 'Device Time: '+'\t\t\t'+reply

message ="AVAIL .\r"
try :
     soc.sendall(message)
except socket.error:
     print 'Send failed'
     sys.exit()

reply = soc.recv(30)
print 'Available Volatile RAM: '+'\t'+reply

message ="PAVAIL .\r"
try :
     soc.sendall(message)
except socket.error:
     print 'Send failed'
```

```
     sys.exit()

reply = soc.recv(30)
print 'Available Battery RAM: '+'\t\t'+reply

message ="0 FileSpaceFree .\r"
try :
     soc.sendall(message)
except socket.error:
     print 'Send failed'
     sys.exit()

reply = soc.recv(30)
print 'Available File RAM: '+'\t\t'+reply

message ="GetSystemTime .\r"
try :
     soc.sendall(message)
except socket.error:
     print 'Send failed'
     sys.exit()

reply = soc.recv(30)
print 'Up Time seconds: '+'\t\t' +reply

message ="ERRORCOUNT@ .\r"
try :
     soc.sendall(message)
except socket.error:
     print 'Send failed'
     sys.exit()

reply = soc.recv(30)
print "Error Count: "+'\t\t\t'+reply

message ="AUTORUN@I .\r"
try :
     soc.sendall(message)
except socket.error:
     print 'Send failed'
     sys.exit()

reply = soc.recv(30)
print 'Autorun Enabled: '+'\t\t'+ reply
```

```
message ="ANY.TASKS?\r"
try :
     soc.sendall(message)
except socket.error:
     print 'Send failed'
     sys.exit()

reply = soc.recv(30)
print 'Charts Running: '+'\t\t'+reply

message ="FILENAME\r"
try :
     soc.sendall(message)
except socket.error:
     print 'Send failed'
     sys.exit()

reply = soc.recv(30)
print 'Strategy Name: '+'\t\t\t'+reply

message ="TIMESTAMP\r"
try :
     soc.sendall(message)
except socket.error:
     print 'Send failed'
     sys.exit()

reply = soc.recv(30)
print 'Strategy Time: '+'\t\t\t'+reply

message ="DATESTAMP\r"
try :
     soc.sendall(message)
except socket.error:
     print 'Send failed'
     sys.exit()

reply = soc.recv(30)
print 'Strategy Date: '+'\t\t\t'+reply

message ="0 SYS.INFO . .\r"
try :
     soc.sendall(message)
except socket.error:
     print 'Send failed'
```

```
          sys.exit()

     reply = soc.recv(30)

     if reply.find('unknown') > 0 :
          print 'Strategy Stored in FLASH: '+'\t'+'Unknown
(requires 8.1 or greater firmware)'
     else :
          if len(reply)>6:
               print 'Strategy Stored in FLASH: '+'\t'+'1'
          else:
               print 'Strategy Stored in FLASH: '+'\t'+'0'



     message ="?EXISTS }
_____IOCONTROL_STRATEGY_ARCHIVE_DOWNLOAD_INT_TABLE_______1 .
DROP\r"
     try :
          soc.sendall(message)
     except socket.error:
          print 'Send failed'
          sys.exit()

     reply = soc.recv(30)
     print 'Strategy Archieved: '+'\t\t'+reply


def cmdupload( soc,file ):
     #Check for the crn files
     if os.path.isfile(file+'.crn1') :
          print 'Ok file '+ file   +'.crn1  found'
     else:
          print 'error: file '+ file    +'.crn1 not found'
          sys.exit()

     if os.path.isfile(file+'.crn2') :
          print 'Ok file '+ file   +'.crn2  found'
     else:
          print 'error: file '+ file    +'.crn2 not found'
          sys.exit()

     if os.path.isfile(file+'.crn3') :
          print 'Ok file '+ file   +'.crn3  found'
     else:
```

```
          print 'error: file '+ file    +'.crn3 not found'
          sys.exit()

     #Check for the chart cdf. files
     fileccd = open(file+'.crn2','r')
     chartlist=[]
     fileline=fileccd.readline()

     while fileline.find('TASK') > 0 :
          #discarding _INIT_IO task
          if fileline.find('_INIT_IO') < 0 :
               start=fileline.find('&')+1
               fileline=fileline[start:]
               end=fileline.find('\r')
               fileline=fileline[:end]

               if os.path.isfile(fileline+'.ccd') :
                    print 'Ok file '+ fileline+'.ccd  found'
               else:
                    print 'error: file '+ fileline    +'.ccd
not found'
                    sys.exit()

               chartlist.append(fileline)
          fileline=fileccd.readline()

     fileccd.close()
     tmstring=time.ctime(os.path.getmtime(chartlist[0]+'.ccd' ))

     timestamp=tmstring[11:19]

     if tmstring[4:7] ==  'Jan' :
          datestamp='01/'
     if tmstring[4:7] ==  'Feb' :
          datestamp='02/'
     if tmstring[4:7] ==  'Mar' :
          datestamp='03/'
     if tmstring[4:7] ==  'Apr' :
          datestamp='04/'
     if tmstring[4:7] ==  'May' :
          datestamp='05/'
     if tmstring[4:7] ==  'Jun' :
          datestamp='06/'
     if tmstring[4:7] ==  'Jul' :
          datestamp='07/'
```

```
     if tmstring[4:7] ==  'Aug' :
          datestamp='08/'
     if tmstring[4:7] ==  'Sep' :
          datestamp='09/'
     if tmstring[4:7] ==  'Oct' :
          datestamp='10/'
     if tmstring[4:7] ==  'Nov' :
          datestamp='11/'
     if tmstring[4:7] ==  'Dec' :
          datestamp='12/'

     datestamp=datestamp+tmstring[8:10]+'/'+tmstring[22:24]
     crcstamp='00112233445566778899AABBCCDDEEFF'

     message ="A\r"
     try :
          soc.sendall(message)
     except socket.error:
          print 'Send failed'
          sys.exit()

     reply = soc.recv(30)

     if reply == '\x00\x00':
          message ="F\r"
          try :
               soc.sendall(message)
          except socket.error:
               print 'Send failed'
               sys.exit()

          reply = soc.recv(30)
          # Don't know  exactly what to expect as response !!!
          if reply == '\x00\x00\x00\x00':
               print ' Invalid response for F command? '
               sys.exit()

     else:
          print ' Invalid response for A command? '
          sys.exit()

     # Skipping  <0 SYS.INFO . . > command because always the
same answer: <SYS.INFO definition unknown> maybe newer hardware
support it!
```

```
     # ****************************
     # Opening transfer session lock ?

     message ="45.0  .\r"
     try :
          soc.sendall(message)
     except socket.error:
          print 'Send failed'
          sys.exit()

     reply = soc.recv(30)
     # Don't know  exactly what to expect as response !!!
Session ID ??
     if reply == '\x00\x00\x00\x00':
          print ' Invalid response for 45.0 AcquireLC .
command? '
          sys.exit()

     message ="A\r"
     try :
          soc.sendall(message)
     except socket.error:
          print 'Send failed'
          sys.exit()
     reply = soc.recv(30)
     if reply <> '\x00\x00':
          print ' Invalid response for A command? '

     # uploading .crn1
     print 'uploading .crn1 file'


     cr1file=open(file+'.crn1', "r")
     cr1line='X'
     while cr1line <> "":
          cr1line=cr1file.readline()
          message=cr1line.rstrip('\r\n')
          message=message+"\r"
          try :
               soc.sendall(message)
          except socket.error:
               print 'Send failed'
               sys.exit()
          reply = soc.recv(30)
          if reply <> '\x00\x00':
```

```
                print ' Invalid response '
                sys.exit()


        cr1file.close()

        # uploading .crn2
        print 'uploading .crn2 file'


        cr2file=open(file+'.crn2', "r")
        cr2line='X'
        while cr2line <> "":
             cr2line=cr2file.readline()
             message=cr2line.rstrip('\r\n')
             message=message+"\r"
             try :
                  soc.sendall(message)
             except socket.error:
                  print 'Send failed'
                  sys.exit()
             reply = soc.recv(30)
             if reply <> '\x00\x00':
                  print ' Invalid response '
                  sys.exit()


        cr2file.close()

        # uploading .ccd task files


        index=0
        while index < len(chartlist) :
             ccfile=open(chartlist[index]+'.ccd', "r")
             ccline='X'
             print 'uploading '+chartlist[index]+'.ccd'+' file'

             while ccline <> "":
                  ccline=ccfile.readline()
                  message=ccline.rstrip('\r\n')

                  # Don't send comments!  Comment lines begin like
this: SPACE BACKSLASH SPACE SPACE
```

```
                if len(message) > 1 :
                     if message[0:4] <> ' \\  ' :

                          message=message+"\r"
                          try :
                               soc.sendall(message)
                          except socket.error:
                               print 'Send failed'
                               sys.exit()
                          reply = soc.recv(30)
                          if reply <> '\x00\x00':
                               print ' Invalid response '
                               sys.exit()

          ccfile.close()
          index=index+1

     # uploading .crn3
     print 'uploading .crn3 file'


     cr3file=open(file+'.crn3', "r")
     cr3line='X'
     while cr3line <> "":
          cr3line=cr3file.readline()
          message=cr3line.rstrip('\r\n')
          message=message+"\r"
          try :
               soc.sendall(message)
          except socket.error:
               print 'Send failed'
               sys.exit()
          reply = soc.recv(30)
          if reply <> '\x00\x00':
               # BUG BUT BUG!! doesn't give problems with SNAP-
UP1-ADS but, gives "duplicate definition with PAC SIM" so ..
               if reply.find('duplicate') < 0 :
                    print ' Invalid response '
                    sys.exit()


     cr3file.close()

     # sending STAMP commands
     print 'sending STAMP commands'
```

```
message=": DATESTAMP .\" "+datestamp+" \" ;" + "\r"
try :
     soc.sendall(message)
except socket.error:
     print 'Send failed'
     sys.exit()

reply = soc.recv(30)
if reply <> '\x00\x00':
     print ' Invalid response to DATESTAMP command '
     sys.exit()
print message

message=": TIMESTAMP .\" "+timestamp+" \" ;" + "\r"
try :
     soc.sendall(message)
except socket.error:
     print 'Send failed'
     sys.exit()

reply = soc.recv(30)
if reply <> '\x00\x00':
     print ' Invalid response to TIMESTAMP command '
     sys.exit()
print message

message=": CRCSTAMP  .\" "+crcstamp+" \" ;"  + "\r"
try :
     soc.sendall(message)
except socket.error:
     print 'Send failed'
     sys.exit()
reply = soc.recv(30)
if reply <> '\x00\x00':
     print ' Invalid response to CRCSTAMP command '
     sys.exit()
print message

# sending FINAL commands
print 'sending FINAL commands'

message="MAKECHECK\r"
try :
     soc.sendall(message)
```

```
except socket.error:
     print 'Send failed'
     sys.exit()
reply = soc.recv(30)
if reply <> '\x00\x00':
     print ' Invalid response to MAKECHECK command '
     sys.exit()
print message

message="CLEAR.BREAKS\r"
try :
     soc.sendall(message)
except socket.error:
     print 'Send failed'
     sys.exit()
reply = soc.recv(30)
if reply <> '\x00\x00':
     print ' Invalid response to CLEAR.BREAKS command '
     sys.exit()
print message

message="[ ABORT\r"
try :
     soc.sendall(message)
except socket.error:
     print 'Send failed'
     sys.exit()
reply = soc.recv(30)
if reply <> '\x00\x00':
     print ' Invalid response to [ ABORT command '
     sys.exit()
print message

#Closing session?
message="ReleaseLC \r"
try :
     soc.sendall(message)
except socket.error:
     print 'Send failed'
     sys.exit()
reply = soc.recv(30)
if reply <> '\x00\x00':
     print ' Invalid response to ReleaseLC command '
     sys.exit()
print message
```

```
     return;


# ********************
#   Main program

if len(sys.argv) > 2:

     if sys.argv[2]=='eraseflash':
          command=sys.argv[2]
     elif sys.argv[2]=='eraseram':
          command=sys.argv[2]
     elif sys.argv[2]=='upload':
          command=sys.argv[2]
     elif sys.argv[2]=='run':
          command=sys.argv[2]
     elif sys.argv[2]=='stop':
          command=sys.argv[2]
     elif sys.argv[2]=='enableauto':
          command=sys.argv[2]
     elif sys.argv[2]=='disableauto':
          command=sys.argv[2]
     elif sys.argv[2]=='storeflash':
          command=sys.argv[2]
     elif sys.argv[2]=='info':
          command=sys.argv[2]
     else:
          print 'unknown  command: \"'+sys.argv[2]+'\" Available
commands: info, eraseflash, eraseram, upload, run, stop,
enableauto, disableauto, storeflash'
          sys.exit()

     try:
          s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
     except socket.error:
          print 'Failed to create socket'
          sys.exit()

     print 'Socket Created'

     remote_ip=sys.argv[1]
     port = 22001;
```

```
     #Connect to remote server
     s.connect((remote_ip , port))
     print 'Socket Connected to '  + remote_ip

     if command=='eraseram':
          cmderaseram( s )

     if command=='eraseflash':
          cmderaseflash( s )

     if command=='info':
          cmdinfo( s )

     if command=='run':
          cmdrun( s )

     if command=='stop':
          cmdstop( s )

     if command=='enableauto':
          cmdenableauto( s )

     if command=='disableauto':
          cmddisableauto( s )

     if command=='storeflash':
          cmdstoreflash( s )

     if command=='upload':
          if len(sys.argv) < 4:
               print 'Upload error, usage: O22term ip_address
upload filename'
          else:
               cmdupload( s, sys.argv[3])

     s.close()

else:
     print 'Usage: O22term  ip_address  command'
```

BLINK.crn1

```
_END NEW $$$.RUN
: FILENAME  ." BLINK " ;
1 0 $VAR *_HSV_SEMA
1024 0 $VAR *_HSV_TEMP
200 0 $VAR *_HSV_INIT_IO
0 IVAR ^_HNV_INIT_IO
```

BLINK.crn2

```
0 TASK  &_INIT_IO
0 TASK &Powerup
0 TASK &slowblink
0 IVAR ^countervar
0 IVAR ^modvar

$0000000000000000.. 32769 1.000000 0.010000 0.000000 2001
$7F000001 0 MBOARD %SELF_RACK
SPOINT 0.0 OUTPUT 0 %SELF_RACK
    0 POINT ~DIGITAL_OUT_1
SPOINT 0.0 OUTPUT 3 %SELF_RACK
    0 POINT ~DIGITAL_OUT_4
```

BLINK.crn3

```
CREATE T.ARRAY
&Powerup ,
&slowblink ,
 0 ,
CREATE V.ARRAY
^countervar ,
^modvar ,
*_HSV_SEMA ,
*_HSV_TEMP ,
 0 ,
CREATE TI.ARRAY
 0 ,
CREATE PTR.ARRAY
 0 ,
CREATE TA.ARRAY
 0 ,
CREATE PTRTABLE.ARRAY
 0 ,
```

```
CREATE B.ARRAY
%SELF_RACK ,
 0 ,
CREATE P.ARRAY
~DIGITAL_OUT_1 ,
~DIGITAL_OUT_4 ,
 0 ,
CREATE PID.ARRAY
 0 ,
CREATE E/R.ARRAY
 0 ,
CREATE E/RGROUP.ARRAY
 0 ,
: CONFIG_PORTS
;
: W_INIT_IO
CONFIG_PORTS
$0000000000000009.. %SELF_RACK ENABLES!
" %SELF_RACK  (1/1)" *_HSV_INIT_IO $MOVE 0 ^_HNV_INIT_IO @!
%SELF_RACK ENABLE
 " Initializing variables" *_HSV_INIT_IO $MOVE
0 ^countervar @!
0 ^modvar @!
 " " *_HSV_INIT_IO $MOVE
;
T: T_INIT_IO
W_INIT_IO
&Powerup START.T DROP
T;
&_INIT_IO ' T_INIT_IO  SETTASK
   : _RUN
   CLEARERRORS
   &_INIT_IO START.T DROP
   ;
```

Powerup.ccd

```
: 0_0

1 LINE.NUM
  &slowblink   START.T
  ^countervar @!

2 LINE.NUM
 \  " working variable to zero!"


3 LINE.NUM
  0
  ^countervar @!
0 JUMP ;
: 0_1

1 LINE.NUM
  ~DIGITAL_OUT_1   ON
0 JUMP ;
: 0_3

1 LINE.NUM
  1000    DELAY

2 LINE.NUM
 \  " This is a comment line!"

0 JUMP ;
: 0_5

1 LINE.NUM
  ~DIGITAL_OUT_1   OFF
0 JUMP ;
: 0_7

1 LINE.NUM
  1000    DELAY
0 JUMP ;
: 0_10

1 LINE.NUM
  ^countervar   1+@!
-5 JUMP ;
```

```
T: T0
DUMMY
0_0
0_1
0_3
0_5
0_7
0_10
T;
&Powerup ' T0 SETTASK
```

slowblink.ccd

```
: 1_0
2 JUMP ;
: 1_3

1 LINE.NUM
  ~DIGITAL_OUT_4   ON
1 JUMP ;
: 1_4

1 LINE.NUM
  ~DIGITAL_OUT_4   OFF
0 JUMP ;
: 1_9

1 LINE.NUM
  ^countervar @@
  5    MOD
  ^modvar @!
0 JUMP ;
: 1_1
TRUE

1 LINE.NUM
  ^modvar @@
  1    <
LAND
IF -4 ELSE -3 THEN JUMP ;
T: T1
DUMMY
1_0
1_3
1_4
```

```
1_9
1_1
T;
&slowblink ' T1 SETTASK
```