# Guide to the iBPP*

March 31, 2001

## 1 Get started

### 1.1 Inputs

Before you start the machine make sure that all boards are seated well. Four kinds of boards and modules exist:

**IF Distribution.** A divider strip in most cases to take the orthogonally polarized IFs and generate up to 6 copies for the baseband Mixer/Filter (MF) and Local Oscillator (LO) modules.

**Mixer/Filter (MF) and Local Oscillator (LO) modules.** These perform analog complex mixing from the IF input band of 300-500 MHz to baseband, 0 frequency, with subsequent low pass filtering by 16,8,4,2,1,0.5 MHz. Gain can be adjusted via an programmable IF attenuator (0-31.5 dB) on the input to bring the output level to $\sim$0 dBm iteratively by inspecting detected power in each MF. There are up to 6 sets of dual MF and single LO units for a total of 12 MFs and 6 LOs. They are housed in the Analog Crate. Monitor & Control are performed via SUN's parallel port link via the Analog Interface Board (AIB).

**Digital Filter Boards (DFBs).** A pair of DFBs is located around each Combiner Board. The DFBs have I and Q analog inputs from the corresponding Mixer/Filter (MF) and Local Oscillator (LO) modules. Pairs

*First drafted by M. Kramer at for Berkeley work–shop in 98dec; guide.tex located at src/pspm/doc in the RCS tree are for orthogonally polarized signals. These perform A/D conversion and digital division (mixing/low pass filtering) into 16(8) SSB(DSB) channels. See 1997 *PASP* **109** 61 for a discussion of SSB/DSB operation.

**Combiner Boards (CBs).** Combiner Boards are located between two DFBs, and will combine and reduce data (sum channel vectors in time, optional sum over polarization) before they are formatted (remove mean and quantize residual in 4 bits) and sent to the SUN via the MB and EDT card[1]. The relation of the data scale to the total power is discussed in a later section.

**Timing Board(TB).** Generates the necessary signals for the DFBs, CBs and MB to observe the pulsar. Requires a 1-PPS signal, a 5-MHz signal, and a connection to the power enable input of the multi-volt Analog Power Supply (APS). Clocks and data transfer rates are discussed in a later section.

**Master Board (MB).** Serves as the interface between the SUN monitor & Control software and the i-BPP hardware via an EDT card which is located in the SUN's S-bus and the VME P1 bus on the i-BPP hardware side. Has reset button and, in some implementations, led's showing communication activity (EDT status and func bits).

Check the hardware table `dat/BPP.HW` for a correct match of the installed boards with its contents and the software table `dat/BPP.SW` for a correct match of settings for current observation. See later section for expansion of the entries in each of these tables.

Besides the inputs described above, there are the connections of the SUN to the backplane of the VME-crate (connectors P1 and P2 of EDT card go to connectors to VME (custom) buses P2 and P3), from the SUN's parallel port to the AIB, and the IF-inputs to the if distribution.

Check `hardware.conf` if you are writing to disk/tape. Check `pulsars.dat` if you are doing timing.

Schematics and related information for all boards and modules of the BPP hardware are located in a 3-volume set of 3-ring binders at each BPP site.

---

[1]Electronic Design Team, Inc. SCD-20 for S-bus at <20 MBs

## 1.2 Login

In order to run the BPP you have to `ssh` into the corresponding SUN computer, i.e. `bkypsr` for LBPP, `pulsarsun` for EBPP, `bcpm1` for BCPM1 and `thira2` for NBPP. For instance,

```
ssh bkypsr.berkeley.edu -l lbpp
```

Only certain domains are allowed to login on each BPP. The list can be found under `/etc/hosts.allow`.

You cannot simply use `telnet` to login into a iBPP and to get a UNIX prompt. Instead, if you do this, you will see the text display with the current status of the iBPP. Hit `Q` to quit. If you really want to enter to the unix shell use port 24 instead, e.g. `telnet bkypsr.berkeley.edu 24`. But if you are observing, you generally do the `ssh`.

The working directory has a different path name for each iBPP. Type `pwd` to find it out. For the LBPP it is `/psr2/lbpp` on `bkypsr`.

The execution binary code can have different locations, depending who updated the code for the backend computer last. Check with `ls -l bin` which programs are used at the moment. If Stuart has, for instance, updated last, then it will point to `bin.sba`. The source code used for compilation will always be under the home directory of the same person but can be either on the host computer or at Caltech on jelly. In the above case, you will find the source is under `/psr2/sba/src/pspm/`. The basic directory structure is the same for all users, *i.e.,* `~/src/pspm`. Under this directory are the various sub-directories for the source code, include files, scripts etc. The master source directory is in Caltech on `jelly.srl.caltech.edu` at `/home/pspm/RCS` which keeps the latest revision of the software.

If for some reason, the ssh-agent is not running on your machine, start it manually, so that compiles and FTPs will work. If the agent is not running on your machine (*e.g.,* on bkypsr or pulsarsun ), you have to execute:

```
ssh-agent tcsh
```

`ssh-add` (this asks for the pass phrase which is the key to entering the secure environment) Copy environment variables.

## 1.3 Launching software

When you have logged in into bkypsr, you will see the following screen:

```
Welcome to the Lab Berkeley Pulsar Processor (LBPP).
```

```
Which of the following do you wish to perform:

  (U)  User Interface          - LBPP interface, implies both (T) and (G)

  (T)  Textual User Interface   - text display monitor for the LBPP
  (G)  Graphical User Interface - graphic display monitor for the LBPP

  (L)  Login                    - normal Unix login

Note: prior to starting the UI or GUI you need to authorize this
      machine to create X11 windows on your local machine by executing
      the following command on your local machine:
xhost +bkypsr.berkeley.edu

Selection :
```

You can ignore the last note, since `ssh` will take care of `xhosts`.

The selection above have the following meaning: If you just want to work on the BPP account, use `l` to get a normal Unix login. In general you type `u` or `g`. Typing `u` you will start the *Status Display*, *Level Monitor*, *Graphical UserInterface* (including the log-message window) and the *Command Line Interface* (see screen capture). Typing `g` will produce all windows except the Command Line Interface. Typing `t` is similar to `g` but will start the User Interface in a separate X-terminal.

You can start these windows also manually from the UNIX login:

```
To manually start the User Interface type: ui
To manually start the Textual User Interface type: tui
To manually start the Graphical User Interface type: gui
```

The Command Line Interface actually runs the main driving routine called *UsrInt*. There can be *only one* UsrInt running per session. If you did not start the Command Line Interface by typing 'u' at the beginning, or `ui` after the UNIX login, you can also start it manually in an existing xterm by just `UsrInt`.

A useful option for command line interaction is *verbose*. More material is spewed out in response to various commands.

You can use the command `ipcs` (inter-process communication facilities status) for running sessions. Another useful command is `lsof [-<pid>]` to check for open devices (specify a PID for devices opened by a certain process).

To capture a dialog of the command line interface, run UsrInt in an xterm: script, UsrInt, commands & replies, exit UsrInt, exit. The result is then stored in file "typescript".

# 2 Graphical User Interface

The Graphical User Interface (gui - usually upper right corner) includes Tkl-buttons plus the window for log-messages. When pressing STATUS, LEVEL or PROFILES you can toggle corresponding windows on/off. By default STATUS and LEVEL are on, while PROFILES is off. For LEVEL you can specify the data to be displayed, while STATUS offers a few choices about time and coordinate system.

Via the PROFILE buttons you can toggle a plot of profiles on/off while selecting the number of channels to be displayed. When you select 1 profile, it should be all channels added together with the DM taken out. The de-disperse button applies to how to plot the number of selected channels, but the DM is always taken out when collapsing more than 1 channel for plotting. So, for example, if you want to look at 4 quadrants of the band you can either have the 4 plot lined up or not depending on the de-disperse button, but the channels that go into each of the 4 plots are always de-dispersed.

The HELP and PAUSE button does not work. Use QUIT to use exit gui including STATUS and LEVEL display.

## 2.1 Status Display

This window (usually upper left corner) will show time, the basic values specified by the user (via UsrInt), the I/O rate through the EDT-card and various coordinate information. Various combinations are possible as set via Graphical User Interface.

## 2.2 Level Monitor

A sample of data is grabbed from ring buffer and statistics can be displayed in real time. Options are: mean channel spectrum; rms about mean (sigma); histogram. The results of single capture (of xxx samples) can be shown, or results can be averaged up to 128 times. A useful sanity check in looking at broadband noise is to set the MF low pass filter to be insufficient for the channel bandwidth selected. Then the 12 passbands of the MFs show clearly. Polarization 0 is left set of channels and polarization 1 is on right.

## 2.3 Profile Monitor

Needs work for iBPPs to allow (a) full number of channels displayed; and (b) fractions of 192 as well as fractions of 128 current. Parameter that is set to determine update time is xxx.

# 3 UsrInt

This is the main control routine which is run by the `tui` part of `ui`, or `U` in a separate xterm window. UsrInt may also be started manually by typing UsrInt. There can be only one per session. There will be a quick warning, if the system detects an already running process. If UsrInt was opened via `ui` then you might be hardly able to read the message, since the window will disappear immdiately. If you suspect, that this a problen, start UsrInt from a normal UNIX-login manually and check-out error messages (see end of last section). If another UsrInt is running and you are sure that you want to take over control, use `pspm_kill` to kill running process before launching software again (see shutdown procedure).

UsrInt has a number of available commands described below which are displayed by typing `help`:

```
accum        display        label        search      tb_reset
aib          dmaprep        leveltest    setif       test
alarm        erase          mbabort      setpower    timing
checklo      exit           monitor      simboard    tstdata
clock        files          open_edt     sleep       utc
close_edt    hardware_conf  power        state       verbose
```

| | | | |
|---|---|---|---|
| `date` | `help` | `powerdown` | `ste` |
| `dfb` | `init` | `printfreq` | `takedata` |
| `disk` | `ioctl` | `quit` | `tape` |

In some cases `help <command>` might work. Find a description of the commands below. (FULL LIST WILL BE AVAILABLE SOON!)

## 3.1 checklo

Reads out the current Voltage Controlled Oscillator (VCO) control voltage values from the LO. These are monotonic in commanded frequency. Use this command to check whether the LOs are set and locked.

## 3.2 files

This commands reads in the current hard- and software table (`BPP.HW` and `BPP.SW`) again. See later section for explanation of lines.

## 3.3 init

If you have connected the machine properly, then you can start to initialize the BPP with `init`. This will initialize the machine and programs analog and digital parts as determined by `BPP.HW` and `BPP.SW`. It will also perform a few tests. Use the `verbose` mode to see more details. Importantly this turns on the Analog Power Supply (unit with batch of leds) if it is not on already.

## 3.4 monitor

The first way to test the correct performance of the machine is to use the `monitor` command, which will enter level monitor mode. Use to view and set the input signal levels.

Monitor will prompt you whether you want direct voltage recording (1=yes, 0=no), will ask for sampling time, single-side-band (SSB) channel bandwidth and if you want to add polarizations (1) or not (0).

`monitor` will start acquiring data without saving them to disk. The level monitor (usually lower left corner) will check if there are valid data in the EDT ring buffers, and will plot them as quickly as it can. By no means is all the data plotted as that requires too much computer power. Selection of integration by >1 integrates adjacent data.

Note that the Combiner Boards packs data into 4 bits by subtracting mean and quantizing noise. To see bits toggle in monitor window with optimum or near optimum rms you need to have exact match between MF bandwidth, MF output power, DFB bandwidth, CB integration time, and CB mode! There is no program to make this easy yet. Normal NBPP operation is:

| | |
|---|---|
| MF bandwidth | : 16e6 Hz |
| MF output power | : 0 dBm |
| DFB channel bandwidth | : 1.5 MHz |

Note that you can change MF settings by `setmf`-command from another UNIX login window. Try `setmf -?` for syntax.

The level monitor has not been taught about 2s complement 4bit numbers Thus, negative values (less than mean after CB summing) are plotted as level 0 to 7; positive (greater than mean) 8-15. We have yet to get this going for direct mode which will display complex DFB directly. In this mode you might have fun playing with the MF attenuation and watching the width of the gaussian distribution vary.

## 3.5 powerdown

After you finished working with the machine, transfer it into a happy sleep state. The command is `powerdown` which will reset boards and shut down Analog Power Supply (led's off). The 5v supply and fan can then be turned off (LAST) if necessary. Power on sequence is the reverse: FIRST 5v, the APS via `init`.

## 3.6 takedata/search

There are two commands which can be used to obtain search data, i.e. `search` and `takedata`. How you need to proceed depends greatly on what type of searching you will be doing.

`takedata` allows the taking of a stretch of search data to disk or tape. It queries the user for the source name and observation length. It does not do any interaction with the telescope at all. Thus it is quick and easy to run once the telescope is pointed at your source and you are getting IF.

Unfortunately, the header it creates may be missing the RA and DEC of the source you observe since the program does not know where the telescope is pointing (in contrast to `search`).

Then, we have a command called `search` which is highly NBPP-specific because it relies on observation schedules downloaded over the serial port from the Nancay telescope computer. This program calls `wptape` with a special option that make `wptape` do a loop over many survey points. Basically it is tied to the schedule program. Each time a new survey pointing in the schedule begins, `wptape` starts datataking with the NBPP, sets the levels, and writes a fixed length file to tape. Then it stops datataking and sits around waiting for the next source. Thus it requires almost no interaction from the operator once it starts (except for changing tapes).

`takedata` just starts data taking in whatever configuration the machine is currently in. Thus, one normally sets `BPP.SW` for the configuration wanted, then run `init` or monitor, then `setpower`, then `takedata`. It asks if you want to write to disk or tape. Then, it uses the values set up in `include.lbpp/machine.h` to decide where to dump the data. It writes disk data to DATADIR which for the nbpp is `/home/thira2/nbpp`. For the ebpp it will write the files to `/pul7/ebpp`, for the lbpp to `/psr2/dat`. It names them with the basename that you give it and adds the scan number to that name. The tape drives which are used by `wptape` are specified in `dat/hardware.conf` on the target machine.

### 3.6.1 Data Format

The data for the PSPM, and BPP machines are very similar. The format is a 32k header, which is defined in `pspm_search_header.h`. After the header, there is just raw data. Each sample is a 4-bit unsigned integer. Each machine has a different number of samples per filter bank vector; the number is 96 for the NBPP. So, each vector is 48 bytes (96 4-bit numbers), and samples are simply written one after another. When the polarizations are not summed the number is double. For the PSPM the order is simple channel order. But, for the BPP machines, the order is much more complex. Look at the order of frequencies in the `*.i`-file (produced by `pspm2psrpack` to see the order. Basically the MB cycles through the the all Combiner Boards reading 4 channels (1W, 2B, 16b) from each CB. Then the register address on each CB is advanced and the MB reads the next words and continues until all are read, 4 when polarizations are summed, 8 when not. The source to `nbpphead` or `pspm2psrpack` might be useful to look at. Also the `gen_chan` routine under RCS.

## 3.7 ste #/utc #

`ste` function programs the STE-register on the Timing Board with the value given in the argument. This is extremely valuable "sanity check" that communication from SUN via EDT to MB and then via VME to TB is ok. Register is read for comparison. Similarly `utc` function programs the long utc counter which is incrementing at 1 PPS (external input) rate. The function reports back the next 3 counts to show that all is ok.

## 3.8 state

Executing this command will give you a condensed summary of the current state of the machine. Note that not all entries are actually read off from the hardware, but as they should have been set from the hard- and software tables (`BPP.HW` and `BPP.SW`), *i.e.,* if some values could not be set, it might not show up in below listing!

```
LBPP : CLOSED> state
cb_mode =  0x06 0x03 0x03 0x03 0x03 0x03
cb_id =  0xB0 0x00 0x00 0x00 0x00 0x00
cb serial =  0x11 0x00 0x00 0x00 0x00 0x00
cb rev =  0x01 0x01 0x01 0x01 0x01 0x01
cb_accum_len = 468

mb_xil_rev 0x02
mb_cb_enable 0x01 mb_rd_error 0x00
mb_adr_end 0x07; mb_vme_low = mb_adr_strt = 0x28
mb_brd_end 0xB0; mb_vme_brd_id = mb_brd_strt = 0xB9
mb_vme_mid_adr 0x00; mb_vme_mid_val 0x00
bpp_sw_init.mb_long_ds0=0x0C; mb_vme_enable=0x0E

tb_ste 0x0000000A; tb_outs_reg 0x00; tb_ins_reg 0x00
Fclk 1.60085e+07; Har_Clk 3.20171e+07; H_deci_factor 16
```

```
DDS_out_freq 4.00213e+06; DFB_DDS_clk_multiplier 16
bandwidth 2.00107e+06
FULLSCALE 171.799; Rpfs 687561271; Hclksel 0x00, harris_dec 15
extra_div 2; Xclksel 0x04; TIM_STATE 4
Tclkstat 0x00, Extstat 0x00
Hclkstat 0x0F, Xclkstat 0x04
GenStat0 0xDD ; GenStat1 0x02 ; Ack_Reg 0x01

dfb_mixer_reg =
x0080 x0080 x0000 x0000 x0000 x0000 x0000 x0000 x0000 x0000 x0000 x0000
dfb_sun_program =
x0000 x0000 x0000 x0000 x0000 x0000 x0000 x0000 x0000 x0000 x0000 x0000
dfb_eprom =
x0001 x0001 x0000 x0000 x0000 x0000 x0000 x0000 x0000 x0000 x0000 x0000
dfb_sram_addr =
x00000 x00000 x00000 x00000 x00000 x00000 x00000 x00000 x00000 x00000
x00000 x0000
dfb_har_addr =
x01F8 x01F8 x0000 x0000 x0000 x0000 x0000 x0000 x0000 x0000 x0000 x0000
DFB SRAM mixer freqs:
-7.00373e+06 -5.00267e+06 -3.0016e+06 -1.00053e+06 1.00053e+06
3.0016e+06 5.0027e+06 7.00373e+06
dfb_clip_adc_neg8 =
x0001 x0001 x0000 x0000 x0000 x0000 x0000 x0000 x0000 x0000 x0000 x0000
ASYMMETRIC =
x0000 x0000 x0000 x0000 x0000 x0000 x0000 x0000 x0000 x0000 x0000 x0000
RVRS_ =
x0000 x0000 x0000 x0000 x0000 x0000 x0000 x0000 x0000 x0000 x0000 x0000
FWD_ =
x0000 x0000 x0000 x0000 x0000 x0000 x0000 x0000 x0000 x0000 x0000 x0000
SHFTEN_ =
x0000 x0000 x0000 x0000 x0000 x0000 x0000 x0000 x0000 x0000 x0000 x0000
DFB HARRIS COEF gain :
1.300000 1.300000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
0.00000 0.000000 0.000000 0.000000
DFB I DC OFFSET :
0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
```

```
0.00000 0.000000 0.000000 0.000000
DFB Q DC OFFSET :
0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
0.00000 0.000000 0.000000 0.000000
MAX_DFB_FREQ 35000000.000000 in Hz
LO Boards : 2.88e+08 3e+08 0 0 0 0
MF Widths [MHz] : 8.0 8.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
MF Atten [dB] : 8.0 9.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
=======================================================
                  FROM BPP_SW_INIT
BRD_STRT = 0xB0
BRD_END  = 0xB0
MB_ADR_STRT = 0x00
ADR_END     = 0x07
DFB SRAM mixer multipliers:
  -7   -5   -3   -1    1    3    5    7
dfb_freq_step_divisor = 2
```

## 3.9 timing

The command `timing` will enter timing mode that creates average pulse profiles. Used to automatically and manually conduct timing observations. When executing it, it will first prompt for the pulsar name. You can either enter a real pulsar name or `demo` or `cal`. When running on real pulsar this operates by fixed frequency sampling of data vectors and summing into memory based on phase calculated using polyco.dat. A counter is updated for normalization.

Tempo is in /opt/pspm/tempo (a symbolic link). The tempo.cfg file is there.

### 3.9.1 demo/cal

Both sub-modes are basically identical. `cal` will dump data files on disk, while `demo` will not. In both cases it will prompt for pulse period, dispersion measure (for on-line de-dispersion), number of bins across pulse period and scan length. Note that the resolution determined by these numbers does have a lower limit set by the transfer rate which in timing mode will be mostly limited by the SUN CPU-power to keep up the folding (*e.g.,* PSPM: 12$\mu$s, dependent on cpu for BPP???). After confirming that the chosen numbers are correct, you will specify SSB channel bandwidth and number of polarization added. The specified values for bandwidth and sampling time should appear on the STATUS display.

In principle, an unreasonable input of bandwidth should be overwritten by the software with the closest allowed choice. (REALLY?) The software should also choose an optimal EPROM mode, which is specified at the beginning through the software table BPP.SW.

It will start scans and repeat them until you enter abort (to stop immediately) or stop (to stop after the scan). Watch the I/O rate status to check the throughput. In cal mode the scans are saved under:

PATH/timing/psrcal/psrcal.*.pspm.

The exact PATH might differ. It is determined during compilation and chosen by the value of the variable DATADIR in the include.<backend>/machine.h file of the valid src/pspm source directory (see above). For the lbpp it is, e.g. /psr2/lbpp/dat/ while for the ebpp is pul7/ebpp.

### 3.9.2 Pulsar name

For the moment, a pulsar entered at the first prompt has to be included in the file

/dat/pulsars.dat:

```
# name    dod  oversample   tc    time     freq    nspan  ncoeff
# ----    ---  ----------   --    ----     ----    -----  ------
1133+16    8      2.34     500     300    1410.0    1500     9
```

dod specifies the number of samples, *i.e.,* $2^{dod=8} = 128$ in the above case. oversample determines the ratio physical sampling rate to the resolution of the final profile given by dod; oversample can be fractional so that fixed bandwidth and CB integration times can be used and allow period to be divided into binary power of bins. The frequency is used for the tempo prediction of the period and does not necessarily agree with the observing frequency specified by the LO-setting! As long as TEMPO11 is not used, the timing model should be included in /tempo/tztot.dat. Since all set-up data are calculated automatically, the scan should start automatically.

(ADD DESCRIPTION OF OTHER PARAMETERS IN pulsars.dat ETC.)
The data will we written to:

PATH/timing/psr<pulsarname>/psr<pulsarname>.*.<backend>.

The exact PATH might differ. It is determined during compilation and chosen by the value of the variable DATADIR in the include.<backend>/machine.h file of the valid src/pspm source directory (see above). For the lbpp it is, e.g. /psr2/lbpp/dat/ while for the ebpp is pul7/ebpp.

### 3.10 verbose

Toggles verbose mode of operation. Switching to verbose mode provides you with much more detailed information during all procedures, in particular init, monitor and timing.

## 4 Setting levels

In order to modify the MF output, you can execute

setmf <#module> <bandwidth> <atten>

from another UNIX-login window. The commands might effect the performance of a running mode, e.g. executing while doing timing the scan might restart again.

The power level should be adjusted automatically by means of the adc-files. This is automatically done in the timing mode (if adc-files installed). Or you can call setpower in UsrInt to set MF attenuation. Note they have chosen to set the MF output power to be approximately 1 dBm which might have been changed this to be 0 dBm.

## 5 Finishing an observing session

When finished type abort or quit any running mode, and powerdown machine. Use QUIT button on gui to close windows. Use exit to leave UsrInt.

If you want to clean-up system completely, you can type pspm_kill on a UNIX-login. DO NOT execute this instruction while disk buffers are still being written to tape after end of data taking. pspm_kill removes all the pspm related shared memory structures from memory. This is good to do after finishing ui, gui or tui as it frees up alot of system resources. It might be helpful to run this command a few times. If you did run it, make sure that before log-off you run (once only!) `pspm_shm` which will restart the shared memory managers (pspm_pre, pspm_mes), so that individual programs can be run after a reboot or `pspm_kill`. This deamon has to run, so that BPP will work. If it is non running and you launch the software, the BPP will be in waiting mode, until this process is existing.

If full crisis, reboot SUN: `su root`; enter password; `/usr/sbin/shutdown -i 6`; this 'graceful' exit settles disk before shutting down. If you want to just shut down and turn off machine, use 5 instead of 6. If you don't know root pwd and must reboot: `stop/a; sync; boot`. The recommended procedure can also be done with: `su root`; enter pwd; `/sbin/init 6`.

# 6 Data reduction

We are in the process of linking iBPP to Dunc Lorimer's SIGPROC package. This will bring a family of modern and versatile programs to the user.

A number of small programs exist to deal with the created data. For most of the you get a description, when you type the program name without an argument.

`pspm2psrpack` Extract the header information and summarizes the file of any mode (including file structure) on standard output. Use `pspm2psrpack file > file.i` to create a `*.i` file.

**timing mode:** `pspmtoflt` Converts a timing-mode file into a binary file with the data backed in four byte real (float) numbers. For each channel all samples are dumped to an output. The channels have been re-orderd into proper frequency order.

`pspmtopgm` Converts a timing-mode file into a pgm-file (portable bit map). Use `xv` to view it.

**search data:** First run `pspm2psrpack` on the search file to create a `*.i`-file (ifile).

`fbstat <file> @ <ifile>` summarized search file including means, rms etc. and produces `*.stat`-file (stat-file).

`statplot` program to plot stat-files. Try `?` for help.

`fb2dd <file> @ <ifile>` Works on original search-file, checks parameter and de-dispersesls data. Produces time series of de-dispersed data (`*.dd`-file).

`ddlook` **¡file.dd¿** `-zap none` plots dd-file.

`dd2pwr` Calculates power spectrum.

`name2txt` looks for search candidate.

# 7 RCS basics

## 7.1 Making an account

```
a) su root              become system administrator (root)
b) edit  /etc/passwd
         /etc/shadow
   mkdir /psr2/dbacker
   chown dbacker /psr2/dbacker
   passwd dbacker
     new ....
     reenter ....
c) su dbacker
   cp ~sba/.login .
   cp ~sba/.cshrc .
d) make alias for make or smake ?
e) mkdir src
   cd src
   mkdir pspm
   cd pspm
   ln -s /home/pspm/RCS RCS
   make |tee make_err           make the system, saving record to file
```

## 7.2 Initialize your directory tree

If you do `not` have installed a copy of the PSPM/BPP-code yet, you can follow the next steps to create your own copy:

1. Goto to your home directory and create the directory `src`, i.e. `mkdir src`.

2. Change into this directory and create `pspm`, i.e. `cd src; mkdir pspm`.

3. Change also in this directory (`cd pspm`).

4. Install a soft links by `ln -s /home/pspm/RCS RCS`.

5. Type `make` which will create the dir-structure. This one often fails. If `smake` not found, check path; you may need to add path to ˜/`src/pspm`. If illegal architecture is problem, add `SARCH=sun5` (or appropriate arch) to environment. Basically you may want to copy the .cshrc from a working account to your new account.

## 7.3 Alter code and test using your directory

```
a) smake -co                        check out all the latest & greatest
b) co -l {path/module}              check out with lock a module;
                                       e.g src/bpp_ut.c
c) edit path/module                 make changes
d) smake {machine}                  download binaries to your bin subdirectory
                                       to see if changes compile;
                                       e.g., smake lbpp. Here you must have
                                       ssh-agent running.
e) ssh to {host cpu}                switch to standard user, basic login
f) ui                               fire up interface and test away...
g) back to c-d-e, continue with change/test
h) either 'ci -u {path/module}'     to store new version, OR

          'co -l {path/module}'     revert back to original
          'ci -u {path/module}'       and store it releasing lock
```

Note that you can swap between your `bin.me` and standard `bin` by `rm bin; ln -s bin.me bin`.

You can delete a file with `rm src/RCS/foo.c,v`, and then your local `foo.c`. This will leave orphaned versions in other remote locations. These will go away when their area is cleared and refilled.

You can backup a version with `rcs -o1.13 src/foo.c`; this deletes latest version, number 1.13.

## 7.4 Summary of RCS commands

When modifying software under RCS control, you have to use the following commands:

**`smake co`.** checks out, with read privilige, from the master source tree all files that are more recent then the local version. To check out means to build a copy of the current revision of the given file. For additional documentation see `man co`.

**`smake colist`.** list all the files in the master source tree that are more recent than the local copies. The more recent files are not actually copied to the local system.

**`smake ci`.** check in all the local files that are more recent than the master source. To check in means to revise the master source file with necessary information so that it can recreate this new revision.

**`smake cilist`.** list all the local files that are more recent than the master source. The more recent files are not actually copied to the master system.

**`smake locklist`.** Gives summary of what files are ci'd.

**`co file`.** build a local read only copy of the lastest revision of this file from the master source. Other than the most recent versions can be created as well. Example: `co include/pspm.h`. See the co man pages.

**`co -l file`.** Similar to co file except that the local file can be written to as well as read. The `-l` (i.e. lock) is to take ownership of this file's RCS lock so that if another user could not perform a `co -l` as well. This means there should always be just one version of each file. Example: `co -l include/pspm.h`

**`ci -u file`.** Check in and revise the master source file with necessary information so that it can recreate this new revision on the local system.

The `-u` means to keep a read-only version on the local system. If this is not included the local system will no longer have a copy of this file. RCS will ask the user to enter information describing the changes in this new version. Typically, this is just a line or two. These comments are stored along with the rest of the file in the master RCS sources. Example: `ci -u include/pspm.h`

`ci -i file.` create new file.

`rcsdiff <file>.` compare the current version of the specified file with the last version checked into the master sources. Example: `rcsdiff src/UsrInt.c` It can be handy to run this command to see what changes you have made before checking in your changes.

`smake <backend> <file>.` perform the make for the specified file for the specified backend but this new object or executable is NOT FTPed to the Sun that controls that backend. This can reduce compile times significantly. Example: `make ebpp UsrInt`

`smake <backend>.` perform the complete make for the specified backend but these new executables are NOT FTPed to the Sun that controls that backend. Example: `smake lbpp`

`smake <backend> update.` perform the complete make for the specified backend and actually FTP the new code to the Sun that controls that backend. It will also change the `bin` link in the backend account to your new, updated bin-directory. Example: `smake lbpp update`

`smake <backend> clean.` Clean up object files etc. It will be necessary after a OS-upgrade.

## 7.5 Creating a new RCS-controlled file - baj 01mar26

Somewhere (probably at top) put in $Log$. If the file is source code, make the line a comment in the syntax of the compiler, so that all log messages will be comments. For example,

```
for latex:
%
```

```
% \$Log\$
%
```

```
for c:
/*
* \$Log\$
*/
```

```
but not
```

```
/* \$Log\$
 * */
```

I think this last one will work but RCS will complain about it; I think it doesn't like having anything after the `$Log$` on the same line.

If you don't want all the log messages to appear in the file but do want basic revision information, use $Header$ instead.

There is a reasonably useful RCS tutorial at [2] but for some reason it doesn't know about the $Log: keyword.

After the includes to the library files, you should enter the following line just before your code-dependent includes:

```
#include "rcsid.h"
```

which is also used by RCS to allow access to version of software in software.

In order to compile your file with `smake <backend> <file>`, it has to be included in the `Makefile`. Change it while watching how other files are treated with their dependencies. Be careful when including backend specific commands. Look how similar cases are handled. It can help to check the new file in before compiling.

When checking in with `ci <file>`, RCS will prompt you for a description of the file. Finish your comments with a dot.

# 8 Principle software components[1]

[2]http://www.nmt.edu/tcc/help/tool/rcs/initial.html

[2]See excellent summary at http://134.104.64.34/pulsar/ebpp/soft/

SUBROUTINES (in `/src`)

`bpp_ut` user level routines for bpp machines

`bpp_edt` A low level routines for bpp machines

`nbpp_takedata` middle level routines for bpp data acquisition

`pspm_fold` workstation realtime data folding

`pspm_sm_monitor` super mongo plotting levels, histograms, ...

`pspm_sm_voltage` will be for voltages

`pspm_util` gets data format from `UPDATE_PARAMS`, unpack rawdata

`pspm_timing` lower level code for UsrInt's get timing data command

`start_auto_timing` UsrInt's timing command to take timing data

PROGRAMS (in `/src`)

`UsrInt` the interactive user interface

and in increasing order of complexity :

`wpnull` take data but don't save it anywhere, is using ring buffers. sets flags to say "just read buf 3", ...

`wpmem` ring buffer for as much memory as workstation has. copies from ring buffer to mem array and when whole mem array is full will stop and copy to disk.

`wpdisk` take data and store to disk until disk fills, no striping, max speed is single disk xfer rate. this is first non-trivial program with threads that wakes up when a buffer is finished. All the threads have the same virtual address space which saves context switch time by large amounts.

`wptape` take data and store to tape 1 thread per buffer. when thread wakes up it writes it to disk (knows about striping). Other threads wake up when they see enough data to send (with unstriping) to the tape. There is one of these threads per tape drive `pspmtime` process all the data in real time - does folding.

The basic flow in the code for the timing mode is, for instance:

`UsrInt.c` ⟶ `start_auto_timing.c` ⟶ `pspm_time.c` ⟶ `pspm_read.c` ⟶ `pspm_fold.c`

In `pspm_read.c` you will see the topocentric time/phase/frequency (fdot hardwired to 0 for now) calculation with a few `#if 0 printf() #endif` sections that may be helpful.

In `pspm_fold.c` you will find the actual folding routine: `fold_fixed_precise()`

# 9 EPROM modes

The CB re-quantized the data via different EPROM modes. The choice of mode will depend on the time resolution (i.e. decimation done on the CB as well decimation done by the software), channel bandwidth and maximum CB, EDT and MB data transfer rates. For the NBPP the EPROM modes are:

| | |
|---|---|
| Mode 0 | Engineering table shft 7 |
| Mode 1 | n= 18, scale= 0.233334 mean= 81.720001, level= 10.453877 min= 13.769798, max= 149.670212 |
| Mode 2 | n= 36, scale= 0.233334 mean= 163.440002, level= 14.784016 min= 67.343903, max= 259.536102 |
| Mode 3 | n= 72, scale= 0.233334 mean= 326.880005, level= 20.907755 min= 190.979599, max= 462.780426 |
| Mode 4 | n= 144, scale= 0.233334 mean= 653.760010, level= 29.568031 min= 461.567810, max= 845.952209 |
| Mode 5 | n= 288, scale= 0.233334 mean= 1307.520020, level= 41.815510 min= 1035.719238, max= 1579.320801 |
| Mode 6 | n= 384, scale= 0.233334 mean= 1743.359985, level= 60.355492 min= 1351.049316, max= 2135.670654 |
| Mode 7 | n= 768, scale= 0.233334 mean= 3486.719971, level= 85.355553 min= 2931.908936, max= 4041.531006 |

A discussed choice for EBPP and BCPM1 was (resolution given for max bandwidth, i.e. 2 MHz in SSB mode, and min bandwidth , i.e. 62.5 kHz in SSB mode):

| | | | res max bw ($\mu$s) | | res min bw ($\mu$s) |
|---|---|---|---|---|---|
| Mode 0 | div by | 1 | 0.25 | * | 16 |
| Mode 1 | div by | 2 | 0.50 | * | 32 |
| Mode 2 | div by | 4 | 1.00 | * | 64 |
| Mode 3 | div by | 8 | 2.00 | * | 128 |
| Mode 4 | div by | 16 | 4.00 | | 256 |
| Mode 5 | div by | 32 | 8.00 | | 512 |
| Mode 6 | div by | 64 | 16.00 | | 1024 |
| Mode 7 | div by | 128 | 32.00 | ** | 2048 |

***** only a subset of channels can be read due to the CB or EDT SCD-20s or MB max transfer rate.

****** Should this mode have a larger division to get to a slower data rate or can we rely on the workstation to provide additional decimation before the data is folded and/or stored to mem, disk or tape ???