Subarray Operation at GMRT

Jayaram N Chengalur

12/Dec/2000

1 Introduction

This writeup documents the multiple subarray version[1] of the DAS software. Up to 8 subarrays can be active at any given time. Two different subarrays can be tracking different sources, have different RF frequencies, and different LTA periods. However they **must** have the same baseband bandwidth. The principal immediate application of subarray operation is to permit simultaneous astronomical observations and maintenance/data quality checking. This current version of DAS is also a step towards a version that would support features that are standard in any astronomical observatory. Although not all these features are implemented yet,it is hoped that they can be done in an incremental fashion.

The various programs that one has to run remains the same as before, viz. **acq** **dassrv** , **acq30** , **sockcmd** ,**collect** , **mon** , **record** . However the commands that these programs accept have changed.

2 So Many Subarrays, So Little Time

As before prior to starting an observation session, one needs to start the programs[2], **dassrv** , **sockcmd** , **acq30** , **collect** . The order in which these programs are started is irrelevant. Once this chain is in place, the programs have to be initialized via the **init** command. This command is now a *privileged* command, i.e. it can be issued only by **ONLINE** user0, which is by convention the on duty telescope operator. Initialization parameters can be passed via the **init.hdr** file as before. **Please note that the parameters in the init.hdr file (eg. the number of channels to record, and the LTA duration) are common to all subarrays! In record however, one can integrate over and above the LTA given in the init.hdr file.** A sampler selection can no longer be made in the init.hdr file. Instead the data for all samplers with valid connections in the corrsys.hdr file will flow from the correlator PC to **collect** on mithun. As described in more detail below, the user can select a subset of this data for monitoring and recording. Since it is not possible to add antennas later in the session, it is important to initialize with all available or potentially available antennas[3].

[1]The documentation is meant for acq30 ver0.98, collect ver1.00, fstop ver0.96 and dassrv ver0.92.

[2]Note that since different software at the GMRT evolves more or less independently, **dassrv** and **sockcmd** have to serve as the infinitely elastic substance that stands between the irresistible **ONLINE** force and the immovable correlator object. As such they perform various contortions to dynamically map between the conventions used by these two different systems, and have hence become, most regrettably, crucial elements in the DAS chain. Should **dassrv** or **sockcmd** crash, all active subarrays will be orphaned and will have to be killed by hand (using **getcmd**), after which the DAS chain will need to be restarted.

[3]DAS also requires a BandMask, i.e. which sidebands of which polarizations to record. This is a bitcode with the first four bits being USB_130, USB_175, LSB_130 & LSB_175 respectively. A BandMask of 3 hence implies record the 130 USB and the 175 USB.

OK. Now having initialized the DAS chain, one needs to start some observation with a specified set of antennas. This is where things get a little hairy. To start an observation you need to first have control over where the antennas will point. For this the **ONLINE** master will give you, the user, some subset of the available antennas. This is the **ONLINE** subarray that you could control, point at your calibrator, or your source, set the wrong LOs for etc. We will refer to it as the **ONLINE** "permitted subarray". **ONLINE** however allows the user to define a subset of the "permitted subarray" as the "(in)use subarray". Commands that are typed by the user at the **ONLINE** prompt are sent only to the the antennas in the "(in)use subarray". So if for whatever reason you wish to exclude an antenna given to you by the **ONLINE** master you can. The subarray that is most familiar to most users is this "(in)use subarray", it is the set of antennas that are displayed in the "subw" window of **ONDISP** . The "permitted subarray" is shown the "gens" window which most users usually do not visit. The **ONLINE** master can dynamically reallocate the antennas among users (hence the "permitted subarray" can dynamically change). So if some antenna is misbehaving, it could in principle be removed from the users permitted subarray and its control could be handed over to the engineer on duty so that repairs and testing can be done. One the antenna is verified as repaired, it could again in principle be given back to the users "permitted subarray".

DAS however has the following constraint. A single lta file cannot contain data for a variable number of antennas. That means that even if an antenna is taken away from the "permitted subarray" one needs to continue to record its data. Further although some antenna may not be available at start the observation, it might become available later. But even if one wishes to use an antenna only at some point in the future one has to start recording its data from the start of the observation. To allow for such situations, DAS allows the user to define a subarray whose data is to be recorded. This is the "recording subarray" and it can be a superset (but **not** a strict subset, see below) of the **ONLINE** "permitted subarray." The upshot is that any user can record the data from any antenna, i.e. even antennas that are not under his/her control. Of course as far as the user is concerend this data is likely to be garbage.

Although it is permitted for one user to record data from antennas that "belong" to another user (i.e. which are in another users **ONLINE** "permitted subarray"), it is important that one and only one fringe stop model be applied to any given antenna. Hence a user can fringe stop[4] only those antennas that are in his/her **ONLINE** ' 'permitted array". In DAS terminology this is called the "model subarray", and it is exactly the same as the **ONLINE** "permitted array". Further the "model subarray" has to be a subset of the "recording subarray" (i.e. you cannot fringe stop antennas that you are not recording) and hence the "recording subarray" must be a superset of the **ONLINE** "permitted subarray". Whew.

Fine. Great. Wonderful. But what does this translate to on the ground, or more accurately, at the kumbh terminal? To find out, take a deep breath, skip the rest of this section, go straight to appendix A. If you continue with this section, you could learn various things about the DAS which may be vaguely useful.

[4] i.e. supply a "model" in DAS terminology

3 DAS Subarrays

Let us start with a breif overview of the DAS. Data from the correlator is read into the PC via a special purpose PCI card, (the "fishcamp" card) by the device driver for that card (pcidev.o). This device driver is automatically started when the correlator PC (Duall) is booted. This very raw data is then re-organized by a higer level program, viz. **acq** . **acq** is also responsible for putting timestamps on the data. Neither the device driver nor **acq** have any idea as to what the correlator setup is, or what antenna is connected to which sampler. They simply collect whatever data is presented to them by the hardware. The next program in the chain is **acq30** . This is the program which understands antenna connectivity (and also subarrays). The antenna connectivity is read from a header file, the "corrsys.hdr" file. **acq30** also computes (separately for each antenna) the required fringe stopping and delay tracking parameters and supplies them to a program called **fstop** whose job is to load these parameters in a time critical fashion into the control cards of the correlator. **acq** and the device driver are essentially free running, they do not accept user commands. User commands are accepted from a message queue by **acq30** . **acq30** does not care from where commands appear on this queue, if it finds a command on the queue it attempts to execute it. There are currently two ways of putting commands on the queue, one via the program **getcmd** , the other via **sockcmd** . **getcmd** is largely meant for debugging. It runs on the corelator PC and passes commands typed on stdin to **acq30** . **sockcmd** is what most users require, it also runs on the correlator PC but it reads commands sent to it across the network by **ONLINE** (via the program **dassrv**) and passes them on to **acq30** . In this section "raw" **acq30** commands are described. This is what you would type to **getcmd** . In appendix A, the **ONLINE** commands that achieve the same thing are listed. **sockcmd** and **dassrv** are responsible for mapping from what **ONLINE** gives to what **acq30** needs.

After having set up the DAS chain, the next step is to define the various subarrays involved. The DAS software identifies each observation ("project") with a **ProjectCode** . The **ProjectCode** is a 7 letter code that intended to be a unique identifier of the project[5]. A project has a "recording subarray" (which is fixed) and a "model subarray" (which can vary with time, provided that it is at all times a subset of the "recording subarray"). Projects are defined using the command **addproj** . This command requires an antenna mask and a band mask. These masks specify the antennas and polarizations/sidebands whose data is to be recorded. **addproj** takes an argument which is the name of a "scan.hdr" file, in which these various masks (and other useful information) is given. See appendix B for an example "scan.hdr" file. Having defined a project, one can then start and stop "scans" on different sources. The corresponding commands are **startscan** and **stopscan** . The **startscan** takes an argument which is the name of a "scan.hdr" file, and the argument to **stopscan** is the **ProjectCode** . The **startscan** command associates a source (and also RF frequencies, etc) as well as a model antenna mask and model bandmask with the project. These masks specify the antennas for which the "model" (i.e. fringe stopping) will be applied.

The first **startproj** command received by **acq30** starts the data flow. The data flow will then continue for as long as the DAS chain is alive. In particular, even if there is no active subarray, the data will continue to flow for all the antennas that were specified in the "corrsys.hdr" file. A number of bookkeeping parameters can be specified in the "scan.hdr" file. These are read by **acq30** and passed on to **collect** . However for backward compatibility with **xtract** and other offline programs not all these parameters them will be recorded into

[5]i.e.it is suitable to use as an identifier key to search a putative archive of observations.

the lta file. When it is possible to update the offline software, these parameters will be recorded into the lta file. The bookeeping parameters include:

1. FLUX: The I, Q, U and V fluxes of the source.
2. MJD0: The epoch of RA_DATE and DEC_DATE.
3. RA_DATE, DEC_DATE: The apparent right ascention and declination of the source.
4. DRA, DDEC: The rate of change of RA and DEC (for solar system objects).
5. FREQ, FIRST_LO, BB_LO: The frequency of the reference channel (i.e. channel 1), the first and fourth LO settings.
6. CH_WIDTH: The channel width[6].
7. ID : The source ID.
8. NET_SIGN: Positive means that sky frequency increases with channel number, negative means that sky frequency decreases with channel number.
9. REST_FREQUENCY: The rest frequency of the line being observed.
10. LSR_VEL: The velocity corresponding to the reference channel frequency.
11. CALCODE: A Calibrator Code.
12. QUAL: Source qualifier (useful for source association in offline analysis).

Data is recorded into a disk file (the "lta" file) using the program **record** . Data can also be monitored online using the program **mon** . **record** and **mon** need to be told which project's data they are to record (or monitor). This is specified via the **ProjectCode** which has to be the first argument to the **record** or **mon** program.

The basic unit of integration of the visibility is the STA cycle, which is about 128 milliseconds. This integration (called Short Term Accumulation) is done in the hardware. The next level of integration (Long Term Integration, LTA) is done by **acq30** . Since this is common to all subarrays, it will presumably generally be left at some small value like 8 STA cycles (i.e. $\sim$ 1 second). However for most projects this is a very short integration time and most observers would like a longer integration in order to keep the data volume low. This further level of integration can be done by **record** . The amount of integration to be done by **record** is specified as a command line option (argument 3 on the command line, see Appendix A) and is in units of the LTA already done by **acq30** . Thus for example if one keeps an LTA of 8 in **acq30** and asks record to do a further 16 integrations before writing to disk, the effective integration time is $16 \times 8 \times 0.128 = 16.4$ seconds.

When all the scans for a given project are over, the project can be deleted with a **delproj** command. This command will cause any **record** (or **mon**) program that is active for this project to exit. After all projects are over, one can kill the complete DAS chain using the **finish** command.

[6]Note that the inter channel spacing need not equal the channel width if for example only every alternate correlator channel is recorded.

Once a project has been defined using the **addproj** command, no other project with the same **ProjectCode** can be defined. Also no project with an overlapping "model subarray" will be accepted. Once a **delproj** command is given however one can define a new project with the same "model subarray" or **ProjectCode** as the old project.

4 Timing, Latencies, etc.

acq30 updates the model (i.e. the fringe parameters) once every ModelCycle, which is currently set to 16 STA cycles. The data for each STA interval is given a unique and monotonically increasing sequence number (DataSeq) by **acq** . The model is updated every time DataSeq is an integer multiple of ModelCycle. It is desirable that each LTA cycle contains an integer number of ModelCycles and that the start of a new LTA interval coincides with the start of a new ModelCycle. It is hence highly recommended that the LTA interval be an integer multiple of the ModelCycle (i.e. **LTA should preferably be an integer multiple of 16**). All programs will of course work even if LTA is not an integer multiple of the ModelCycle, or even if LTA is less than the ModelCycle. In particular setting LTA=1 is allowed, although if you also record all channels with LTA=1 be prepared for very large data sets. After initialization **acq30** waits for a "strtndasc" command before it starts acquiring data. The data is acquired starting with the first encountered DataSeq that is an integer multiple of LTA. Thus there could be a delay of up to 1 LTA cycle between the *first* instance of a "strtndasc" command and the actual start of data acquisition by **acq30** .

Commands issued from **ONLINE** are sent across the network to the Correlator PC, where there are placed on a message queue by **sockcmd** . This queue is polled for fresh messages every 40 ms by **acq30** . **acq30** executes commands as soon as it reads them. The delay between issuing a command from **ONLINE** to its being executed by **acq30** is dominated by the polling interval in **acq30** , and is in general small compared to the LTA interval.

Once **acq30** executes a command, the information about the change in state is communicated across the network to **collect** . This information is sent asynchronously, i.e., **collect** generally receives the information about the change of state before the data affected by this change of state reaches it. **collect** reflects this change of state in its shared memory (which is accessed by **record** and **mon**) only after the data affected by this change reaches it. In the meantime the information is placed in an "event queue". Thus for example, when **ONLINE** issues a "strtndasc" command, **collect** will queue this request until the Correlator data corresponding to times later than the start command reaches it. In the meantime **collect** prints an informational message

`* NewScnReq Prj prjcode (ScanTabId=n)`

so that the user knows that this command has been received and is pending. Since **collect** waits for the affected data to reach it, it effectively executes commands only at LTA boundaries. When **collect** actually reflects the new state in its shared memory, another informational message

`* ScanStart Prj prjcode (ScanTabId=n)`

is issued to let the user know that the data itself has arrived and that **record** should start recording etc. The length of time that a command stays on the queue depends on the pipeline delay between **acq30** and **collect** . The command queue is 8 commands long, the

DAS chain will collapse if more than 8 commands are pending[7].

record prints an "am alive" message once every record interval (LTA2 x LTA), so if LTA2 is large there can be a long wait between **collect** announcing that a new scan has been started and the user seeing output from **record** . **record** has started recording from the first good LTA interval though, so no data is actually lost.

[7] A queue length of 8 commands is probably generous, but I am willing to be surprised.

5 Appendix A

This is the sequence of operations to be followed when using the subarray DAS software.

1	**duall**	% **corr_config** 1	setup parameters are read from the file "cmd.file' in the directory pointed to by the environmen variable "ACQ_DIR".
2	**duall**	% **acq**	
3	**duall**	% **acq30** , **sockcmd** , **fstop**	The order is irrelevant. The sampler connectivity as well as the baseband bandwidth (CLK_SEL is read from the file "corrsys.hdr" in the di rectory pointed to by the environment variabl "ACQ_DIR". The antenna co-ordinates etc. ar read from the file "antsys.hdr" in the direc tory pointed to by the environment variabl "SYS_DIR". Users do not have write permissio for this file.
4	**mithun**	% **collect**	order unimportant, can be run before (3).
5	**chitra**	% **dassrv**	order unimportant, can be run before (3) or (4) However, should **dassrv** die during the observa tion, all subarrays will be orphaned, and will hav to be killed by hand using **getcmd** .
6	**ONLINE**	> initndas '/t/init.hdr'	This is a **privileged** command, i.e. can only b executed by user0. Unlike previously, there i no sampler selection possible in the init.hdr file The parameter is the absolute path name of th init.hdr file
7	**ONLINE**	> prjtitle 'title'	Title of the project. Up to 80 characters. Use command.
8	**ONLINE**	> prjobs 'observer'	Name of the observer. Up to 8 characters. Use command.
9	**ONLINE**	> tpa r1 r2 1L1 1L2 4L1 4L2 ...	Observing frequency etc. User command[a].
10	**ONLINE**	> prjfre	Copy project parameters into **ONLINE** share memory. User command.
11	**ONLINE**	> lnkndas	Setup the link between **ONLINE** and DAS. Use Command[b]
12	**ONLINE**	> ante N a1 a2 ...aN	Antennas to be recorded. User command
13	**ONLINE**	> initprj 'CODE'	Initialize the project with given CODE. Th CODE can be upto 7 characters long. **Note tha ONLINE maps all characters into uppe case.** User Command.
14	**ONLINE**	> gts 'source'	Get source parameters. User command.
15	**ONLINE**	> sndsacsrc(1,Nh)	Track given source. User command.

[a] Note that by default the bandmask is set to 3, i.e. 130_USB and 175_USB, to use 130_LSB and 175_LSB, set tpa(11) to '0xc'.

[b] This has to be done only once per **ONLINE** session.

16	**mithun**	% **record** CODE file.lta [LTA2]	Start recording data for the project CODE. **Note that ONLINE maps all characters into upper case.** The optional parameter LTA2 is how much integration to do in units of the LTA1 already done by **acq30** . So for example if you give 16 in the init.hdr file and 8 here, then the data in the recorded lta file has an effective integration of $8 \times 16 = 128$ STA cycles. If you set LTA2 to #m, then **record** will compute the median instead of the mean of the # LTA1 records. This is desireable in certain instances (primarily when there are a few extreme outliers.)
17	**mithun**	% **mon** CODE [options]	Monitor data for the project CODE. **Note:** (1) mon is usually easier to use via the TCL/TK script **mon.tcl**. (2) mon has two particularly useful options for weak sources, viz. "-a" to average over channels and "-l" to do a specified amount of LTA2, i.e. integration over and above that done by **acq30** .
18	**ONLINE**	> strtndasc	Start scan on the source got by the last gtsrc. Fringe stopping is done for all antennas that are in the **ONLINE** "permitted" subarray (i.e. those antennas that you see in the **ONDISP** gens subwindow under the title SUBN PERM, where N is your **ONLINE** subarray number. Note that this is a superset of the antennas that you see in the **ONDISP** subarray window (subw) or under the title USRN USIN in the gens window. Frequencies can be changed from scan to scan by setting tpa and running prjfre. User command.
19	**mithun**	% **dasstat** -v	useful diagnostic program that shows the antennas being recorded, the source being tracked, the LO settings being used etc.
20	**ONLINE**	> stpndasc	Stop scan on source. User command.
21	**ONLINE**	> stpprj	Stop the project, also causes record and mon (corresponding to this users' CODE) to exit. User command.
22	**ONLINE**	> hltndass	Stop the DAS session. This is a **privileged** command and can be executed by user0 only.

6 Appendix B

Sample scan.hdr file for getcmd.

```
{ SubArray0.def
ANTMASK = 3fffffff
BANDMASK= 0003
SEQ      = 0
INTEG    = 0.396290
DATE-OBS= Sun Feb 28 00:00:00 1999
OBSERVER=  me
PROJECT =  secret
CODE     = fullcod
OBJECT   = 3C468.1B
RA-DATE = 357.724972
DEC-DATE= 64.668272
MJD_REF = 51236.770833
DRA/DT   = 0.000000
DDEC/DT = 0.000000
F_STEP   = 125000.000000
RF       = 325000000  325000000
FIRST_LO= 255000000 255000000
BB_LO    = 70000000 70000000
} SubArray0
```

Appendix C (Diagnostic Utilities)

The subarray DAS appears to be stable. Most of the problems (not surprisingly) occur because of confusion between which subarray is relevant for a given purpose. Please read the section on Subarrays for the hairy details on subarrays. There are also several utilities to determine the current status of DAS. **ONDISP** has new sub-window, called dasw which gives the status of all the defined projects. Similarly there is a program called **dasstat** which gives a list of all currently active projects. **dasstat** has a particularly useful option "-v" (verbose) which gives the antennas in each subarray, as well as the source that is being tracked, the first and baseband LO settings etc. This can on occassion be a reassuring thing to check.

record and **mon** will wait for the next record block before starting, this could be a long wait if you have a large LTA. If **record** does not find the specified **ProjectCode** it the list of active projects, it will sit there patiently waiting for the **ProjectCode** to appear. This could be a finite wait if you intend to start a scan for that **ProjectCode** sometime soon, or it could be for ever if you have made a typo in the **ProjectCode** supplied to **record** . The common mistake is to forget that **ONLINE** maps everything into upper case. When **record** is waiting, it types the message : `''* Waiting for ScanStart''`