VLBA Sensitivity Upgrade MEMO 41

X-cube Soft Switch Command Set

Walter Brisken

26 July 2013

1 Introduction

This document describes the functionality of the X-cube software switch running NRAO's own `soft_switch` software. The primary purpose of this program is to route data from one or more data sources into one or more recorders. The Mark5C recorder is specifically targetted with this program and hence output pacing is implemented to prevent the Mark5C input interface from being overloaded. Additional features implemented in this software include burst mode (ability to trickle data to the recorder at a rate slower than it is being received for short bursts of time), the ability to write data directly to the X-cube system disk for immediate analysis, and the ability to resend the data over a UDP socket to other software that may want real-time inspection of the data (such as for pulse cal extraction).

2 About the X-cube software switch

X-cube Research and Development Coorporation marketted for NRAO the "Ethernet Software Switch Recorder" product which is a spin-off of their Logger hardware/software. The hardware is an Intel CPU motherboard with 16 GB of RAM, a modest operating system disk and two dual-10 GbE network cards interfaces. The initial intention was to make use of this hardware and a modified version of their Logger software to perform the merging of data streams from the two ROACH Digital Back-Ends (RDBEs) and one or two Mark5C recorders. The X-cube software exhibited one significant problem that is being worked out: the units hang after about 10 recording scans. It is hoped that the X-cube software will ultimately be used. In the interim a simple switching program has been developed at NRAO. It is this software, `soft_switch` that is described by this memo.

The source code for `soft_switch` can be found in NRAO's subversion repository at https://svn.aoc.nrao.edu/repos/VLBA/soft_switch .

This document is current with `soft_switch` version 0.4.

3 Starting `soft_switch`

The `soft_switch` program can be started from the command line.

> Usage: `soft_switch` [*options*]
>
> *options* can be:
>
> > `-h` or `--help` : print usage information and exit
> >
> > `-i` *devices* or `--input` *devices* : set network devices *devices* as inputs
> >
> > `-o` *devices* or `--output` *devices* : set network devices *devices* as outputs
> >
> > `-b` *size* or `--buffersize` *size* : set each input's burst mode buffer to be *size* bytes
> >
> > `-v` or `--verbose` : print more verbose logging/debug info
> >
> > `-q` or `--quiet` : print less verbose logging/debug info
>
> Example: `soft_switch -i eth2 eth3 -o eth4 eth5 -b 5000000000`

The input devices to be used while `soft_switch` is running must be provided on the command line. The output devices specified on the command line are set up to run by default when the program starts but these devices and the details of routing between inputs and outputs can be changed. Upon start all specified input ports are routed to all specified output ports.

When started `soft_switch` will quickly begin its job of switching packets. Stopping it, reconfiguring it, and getting statistics can be done through the VSI-S interface documented below. Log data is sent to the console (both stderr and stdout). See section below on `logsoft_switch` for information on a convenient program to timestamp and write this log data to files.

4 The VSI-S interface

Similar to a Mark5 recorder, `soft_switch` exposes a control interface via TCP port 2620. The command syntax is identical to that used by the Mark5 series of recorders (see http://www.haystack.mit.edu/tech/vlbi/mark5/docs/command5a.pdf for details). By default a maximum of 8 TCP connections to this port are allowed at any one time. Both the port and the maximum number of connections can be easily changed by modification of the source code. The commands and queries supported by `soft_switch` are documented in this section.

4.1 buffer1 – Query interface 1 buffer state

Query: ⟶ `buffer1?;`
⟵ `!buffer1?` <return code> : <device> : <buffer size> : <read position> : <write position> : <overflow> ;

Purpose: View usage of burst mode buffer for interface 1.

Monitor-only parameters:

parameter	type	values	comments
<device>	char	`eth2`, `eth3`, ...	the OS name of interface 1
<buffer size>	int	≥ 0	size of the ring buffer in bytes
<read position>	int	≥ 0	position of read pointer in buffer
<write position>	int	≥ 0	position of write pointer in buffer
<overflow>	int	0, 1	overflow status (0 = ok, 1 = overflow)

Notes:

1. <read position> and <write position> range from 0 to <buffer size> −1.
2. Once an overflow condition is reached operation no more data will enter the buffer until it gets reset at the start of the next scan; data remaining in the buffer will be drained.

4.2 buffer2 – Query interface 2 buffer state

Same as query `buffer1`, but applies to interface 2.

4.3 burst – Configure burst mode

Command: ⟶ `burst=` <A> : <B> ;
⟵ `!burst=` <return code> ;
Query: ⟶ `burst?;`
⟵ `!burst?` <return code> : <A> : <B> :

Purpose: Configure / query the burst mode parameters

Settable parameters:

parameter	type	values	comments
<A>	int	≥ 0	the A parameter of rational burst mode
<B>	int	≥ 0	the B parameter of rational burst mode

Notes:

1. If A or B is zero, burst mode is disabled.
2. The command form cannot be used while `soft_switch` is running but the query can be performed.
3. Burst mode is implemented here as "synchronous rational bursting". In this situation the data entering each interface serves also as the data sending clock. All received data is instantly added to the head of a ring buffer. In a period spanning B incoming packets, A packets are transmitted out. This leads to a very predictable output data rate provided the input data rate is known. If S is the size (say, in bytes) of the burst mode buffer, and R is the incoming data rate (in bytes per second), then the time in seconds before overflow occurs is $\frac{S}{R}\frac{B}{B-A}$. After overflow data can continue to flow from the buffer for $\frac{S}{R}$ seconds. The sum of these two numbers, $\frac{S}{R}\frac{2B-A}{B-A}$, represents the longest useful burst mode scan.

4.4 log_state – Log the state of the switch (command only)

Command: ⟶ `log_state=;`
⟵ `!log_state=` <return code> `;`

Purpose: Mainly diagnostic: sends a description of the state of `soft_switch` to stdout for logging.

Notes:

1. There are no parameters to this command.

4.5 mode – Set mode of operation (dummy)

Command: ⟶ `mode=` <mode> `:` <submode> `;`
⟵ `!mode=` <return code> `;`
Query: ⟶ `mode?;`
⟵ `!mode?` <return code> `:` <mode> `:` <submode> `:`

Purpose: Compatibility with `x3vsi`.

Settable parameters:

parameter	type	values	comments
<mode>	int	≥ 0	
<submode>	int or char	≥ 0 or `null`	`null` implies 0

Notes:

1. The query response for <submode> is always an integer.
2. The command form cannot be used while `soft_switch` is running but the query can be performed.
3. The <mode> and <submode> values have no impact; this command is present only for compatibility with `x3vsi`. The query responses will echo set values.

4.6 pace – Configure pacing of output interfaces

Command: ⟶ `pace=` <input device> `:` <pace> `;`
⟵ `!pace=` <return code> `;`
Query: ⟶ `pace?;`
⟵ `!pace?` <return code> `:` <input device> `:` <pace> `...:`

Purpose: Set hardware pace register on output interface.

Settable parameters:

parameter	type	values	comments
<input device>	int or char	≥ 0 dev name	an integer X is shorthand for `eth`X
<pace>	int	0 to 10	corresponds to # Gbps output rate

Notes:

1. The turn-on default <pace> value for all output interfaces, and the default value when instantiating new routes, is 3, which is shown to work well with Mark5C.
2. The command form cannot be used while `soft_switch` is running but the query can be performed.
3. The query reports on the value of <pace> for all output interfaces.
4. This can only be performed on output ethernet interfaces using the ixgbe driver with the Pause and Pace (PAP) patch (see http://www.zyztematik.org/?p=75http://www.zyztematik.org/?p=75).

4.7 packet – Configure packet filtering

Command: ⟶ `packet=` <input device> : <payload size> : <offset bytes> ;
⟵ `!packet=` <return code> ;
Query: ⟶ `packet?;`
⟵ `!packet?` <return code> : <input device> : <payload size> : <offset bytes> . . . :

Purpose: Set hardware pace register on output interface.

Settable parameters:

parameter	type	values	comments
<input device>	int or char	≥ 0 dev name	an integer X is shorthand for `eth`X
<payload size>	int	unset or > 0	
<offset bytes>	int	≥ 0	number of bytes preceding the payload data

Notes:

1. Leaving <payload size> unset prevents length filtering; this is the default when turned on.
2. The turn-on default <offset bytes> value for all output interfaces, and the default value when instantiating new routes, is 28, which is appropriate for VDIF.
3. When saving data to a file on the system disk, <offset bytes> are removed from the beginning of each packet.
4. The command form cannot be used while `soft_switch` is running but the query can be performed.
5. The query reports on the value of <packet> for all output interfaces.

4.8 reset_stats – Reset packet counters

Command: ⟶ `reset_stats=;`
⟵ `!reset_stats=` <return code> ;

Purpose: Reset packet/byte counters reported with `stats1` and `stats2`

Notes:

1. There are no parameters to this command.

4.9 route – Configure data routes

Command: ⟶ `route=` <input device> : <output device> ;
⟵ `!run=` <return code> ;
⟶ `route=` <input device> : <directory> ;
⟵ `!run=` <return code> ;
⟶ `route= clear;`
⟵ `!run=` <return code> ;
Query: ⟶ `run?;`
⟵ `!run?` <return code> [<input device> : [<output device> | <directory>] . . .] ;

Purpose: Configure or query data routing.

Settable parameters:

parameter	type	values	comments
<input device>	int or char	≥ 0 dev name	an integer X is shorthand for `eth`X
<output device>	int or char	≥ 0 dev name	an integer X is shorthand for `eth`X
<directory>	char		output directory on system disk

Notes:

1. The command form cannot be used while `soft_switch` is running but the query can be performed.
2. The `route` command should be run separately for each data path desired; route commands are additive.

3. `route=clear;` is needed to completely unroute the switch before constructing an arbitray new routing.

4. Unlike `x3vsi`, the routes are not cleared when `run=0;` is commanded.

5. One input device may be routed to zero or more outputs, including output ethernet devices or a file directory.

6. The query reports in input/output pairs all established routes.

7. Only one directory can be associated with each input device. When writing to a directory, systematic filenames of the form *device_unix time* are automatically created. At each gap in incoming data longer than 0.1 second a new output file will be created. If the destination directory does not exist, it is created at configuration time. The output directory can be changed without `route=clear;` but it cannot be completely unset.

8. Judgement must be used when directing data to the system disk. If the disk cannot keep up with the input rate that interface will drop packets.

4.10 run – Enable/disable operation

Command: ⟶ `run=` <state> `;`
⟵ `!run=` <return code> `;`
Query: ⟶ `run?;`
⟵ `!run?` <return code> `:` <state> `:`

Purpose: Enable or disable the switching operation

Settable parameters:

parameter	type	values	comments
<state>	int or char	0, 1, `off`, or `on`	0 or `off` turn off

Notes:

1. The query response for <state> is always either 0 or 1.

2. No configuration of the switch can be performed while running.

4.11 stats1 – Get statistics from interface 1

Query: ⟶ `stats1?;`
⟵ `!stats1?` <return code> `:` <packet count> `:  0 :` <byte count> `:  0 :  0 :  0 :  0 ;`

Purpose: Get statistics for input interface 1.

Monitor-only parameters:

parameter	type	value	comments
<packet count>	char	≥ 0	number of packets received since last reset
<byte count >	char	≥ 0	number of bytes received since last reset

Notes:

1. The 5 extra zeros in the output are for compatibility with `x3vsi`.

4.12 stats2 – Get statistics from interface 2

Same as query `stats1`, but applies to interface 2.

4.13 version – Get the program version

Query: ⟶ `version?;`
⟵ `!version?` <return code> `:` <version number> `:` <program name> `;`

Purpose: Determine software version number and type.

Monitor-only parameters:

parameter	type	comments
<version number>	char	usually of the form X.Y
<program name>	char	always returns `soft_switch`

5 Logging program

Program `logsoft_switch` launches and logs data from `soft_switch`. It is convenient to put this program in one of the operating system's boot-time init scripts so it starts automatically.

Usage: `logsoft_switch` [*options*] [*wait*]

options can be:

`-h` or `--help` : print usage information and exit

`-v` or `--verbose` : print more verbose logging/debug info

`-q` or `--quiet` : print less verbose logging/debug info

`-t` or `--test` : use in developer's test mode

`-a` *a* or `--maxage` *a* : delete log files older than *a* days

wait is a time in seconds to wait before spawning soft_switch

Example: `logsoft_switch 3`

This program starts `soft_switch` and writes log data with timestamps. A new log file is created at the beginning of each UTC day. The log files are given names: `/home/xcadmin/logs/<year>_`*doy_time*`.soft_switch.log` where *year* is the 4 digit year, *doy* is the day of year (1 to 366), and *time* is the UT time of day when the log started.

This program makes use of a state file, `/var/spool/xcube/logsoft_switch.watch`, to determine if the program should be started or not. If the first line starts with `1`, then it is started; otherwise it is stopped. `logsoft_switch` monitors this file and if the file is changed, it will reread the state (0 or 1) and will start or stop the program as needed. Three convenience scripts are available to assist with this:

- `StartSS` : Will start `soft_switch` if not already running
- `StopSS` : Will stop `soft_switch` if not already stopped
- `RestartSS` : Equivalent to running `StopSS` followed by `StartSS`