r e p o r t

MIT Kavli Institute
1 Hampshire Street
Cambridge MA 02139–4307
Tel: 617-253-7277
Fax: 617-253-8084

To:	ACIS Science Operations Team
From:	Peter Ford, NE80-6071 <pgf@space.mit.edu>
Date:	May 17th 2010
Subject:	Flushing a BEP Command FIFO

Introduction

In the early hours of April 7 2010, a new flight software patch load (version 48) was uplinked to ACIS via realtime commands. The relevant command procedure was `SOP_ACIS_SW_STDEOPTE`, and the FOT command script was `I_ACIS_SW_STDEOPTE.ssc`. The loading started at 11:15:56 UT and proceeded without a hitch until 11:23:10 when there was no response to an *addPatch* command within the *eventHist* patch sequence. ACIS continued to report software and DEA housekeeping packets to telemetry, and there was no indication that the *addPatch* had been received. Then we noticed that the `1STAT7ST` flag was asserted in ACIS bilevels, indicating that the BEP's input FIFO wasn't empty. We concluded that ACIS had been sent only the first part of the *addPatch* command, and was waiting for the rest. Flight operations confirmed that there had been a brief telemetry outage at the time the *addPatch* had been uplinked, due to a change-over between telemetry receivers. Since it is not unusual for realtime commands to span uplinked telemetry blocks, a truncated ACIS command would be a natural result. We had been lucky not to have experienced one in previous uplink commanding sessions.

On April 7th, with a partial patch load in the instrument and loss-of-contact approaching, we decided that the only way to clear the FIFO and resume patching was to reboot the BEP. This we did at 11:48:50 UT, and seeing that the instrument reported that its software version was 11, we assumed that it had cold-booted and that the focal plane temperature set-point had reset from –121C to –120C. However, the initial *bepStartupMessage* showed that the BEP had actually been warm-booted, and we realized to our horror that the BEP had only applied the partial patch load that had been uplinked before the FIFO had stuck. We quickly went ahead and sent a *removePatches* command to delete the partial patch load, uploaded a full set, dumped and verified them, and warm booted, successfully, at 12:10:52 UT.

Later that week, analyzing the April 7 problems, we realized that our first reboot had been unadvisable because, by applying a partial patch load, the BEP might easily have crashed—*e.g.*, from an addressing exception or execution of an illegal instruction—which would have taken longer to diagnose than was available during the remainder of the realtime contact. Had we instead cold-booted the BEP, we would have avoided this possibility, but we would have left the focal plane at the wrong temperature since no CAP had been approved to send the commands necessary to change its set point. Better would have been either (a) to have a way of asking the OCC to repeat the dropped uplink telemetry block, or (b) to have developed special ACIS commands to unblock the BEP's FIFO while it was waiting to read the remainder of a truncated command. Being assured by the OCC that (a) won't happen, we concentrated on (b) and, in this report, present a simple solution.

The DPA Command FIFO

The DPA hardware command FIFO consists of a series of registers in each BEP's FPGA that acts as an interface between a serial input channel from the SI-RCTU and a 32-bit address (`0xA0180014`) in BEP processor address space. The FPGA also maintains five bits in the BEP status register (`0xA0180004`) to report on the status of the FIFO, and provides a bit (bit 13, aka `FIFORST`) in the BEP pulse register (`0xA0180008`) to flush and reset the FIFO.

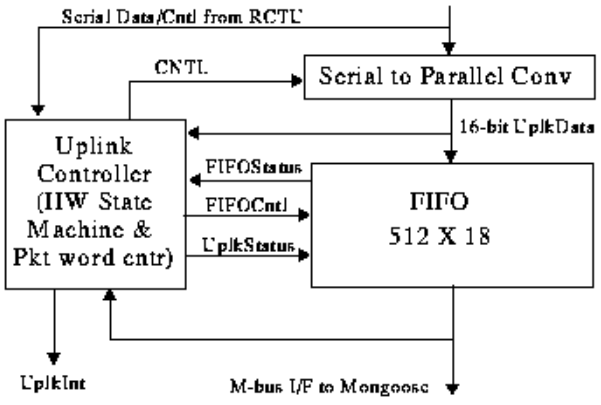
The 32-bit values read from the FIFO contain 2 bits of status information along with 16 bits of data: the `PFW` bit reports the "Packet First Word" flag; if this bit is set, this is a new packet and the 16-bit data value represents the packet length in 16-bit words; the `FNE` bit contains the "FIFO Not Empty" flag, indicating that the 16-bit data is valid. If a FIFO is "stuck", it means that the `FNE` bit continues to be de-asserted before the full set of 16-bit words has been read.

Bit	Hardware Mnemonic	IPCL Mnemonic	Description
0	`UPLKPKTLNERR`		Set when Uplk Controller detects a packet length error; also creates UplkInt (see below). This bit is cleared when the CPU pulses ClrUplkInt (BEP Pulse Register).
1	`UPLKFFULLERR`		Set when Uplk Controller detects a FIFO full Condition and Uplink is enabled; also creates `UPLKINT` (see below). This bit is cleared when the CPU pulses ClrUplkInt.
2	`UPLKINT`		Set when Uplk Controller detects an end-of-packet, a packet length error, or a FIFO full error (see above). The occurrence of either of two error conditions also automatically clears the `UPLKENB` bit in the BEP Control Register. `UPLK-INT` is cleared by writing to the BEP Pulse Register (and is fed directly to a Mongoose external interrupt (see Section 2.1.2.3.5).
3	`UPLKFFULL`	`1STAT6ST`	FIFO Full Flag direct from Uplink FIFO (active low).
4	`UPLKFEMP`	`1STAT7ST`	FIFO Empty Flag direct from Uplink FIFO (active low).
20	`CMDRST`	`1STAT5ST`	Reset Status Bit = 1 if last reset was initiated by the HW uplink port, 0 other-wise. The processor should explicitly reset this bit (via the BEP pulse register) after reading the status word.
21	`BEP_ID`	`1STAT4ST`	BEP ID set via backplane slot (0=A, 1=B)

The operation of the FIFO is explained in more detail in Section 2.1.2.7.1.1 of the DPA Specification (Rev. C) relating to the DPA Command FIFO, and reads as follows:

The peak command rate output by both links is interface limited to 8K-words/sec (a command word is 16-bits). Due to uplink bandwidth constraints, the average command rate is much lower (about 40 commands/sec.). The stored command system may output a series of commands continuously at the peak rate. A synchronous, clocked data line is provided by the RCTU for each serial link; there is no handshaking, and the RCTU is unaware of whether the DPA received a command. See the RCTU Users' Guide for further detail.

The Flight Software overlays a high-level packet format onto the 16-bit words output by the RCTU. BEP hardware provides buffering in the form of an "intelligent" FIFO interface (see Figure 4). The hardware performs a serial to parallel conversion. Upon accumulation of a 16-bit command word, it writes it into the 512 word FIFO (Two RAD-HARD GEC MA7001 FIFOs). At startup the FIFO interface will be enabled and subsequently drained by the flight software. The flight software will at this time determine (via a specified delay implemented by the stored command system) that the first word of a "packet" will be the next word to arrive. It then primes the HW circuitry by setting the `UPLKENB` bit in the BEP control register. The Uplink Controller then looks at the next word, strips out a length field and loads a down-counter with the appropriate packet length. The counter must be decremented upon loading the length, as the packet length count includes the length word itself. The



The uplink interrupt will be latched and reset by the processor during the interrupt service routine. If an ille-gal packet length (3 words < packet size < 256 words) is detected, the HW generates an `UPLKINT` and sets the `UPLKPKTLNERR` status bit. Since the GEC FIFOs are X 9, two extra bits are available and may be used by the HW. Bit 16 is used to mark the HW detected first-word of packet. The FIFO and Uplink Controller Control/ Status bits are memory mapped onto the processor M-bus.

Reads from an empty FIFO are prohibited. The FIFO empty-pulse is explicitly sampled by the BEP hardware and used to gate the FIFO_Read strobe. The FIFO empty flag is available to the processor both as the MDAT MSB (bit 31) of a FIFO read and as a bit in the BEP Status Register. A FIFO-FULL condition is degenerate (i.e. it should never happen during normal operation) and will block any further writes to the FIFO. If the FIFO fills when uplink is enabled, an `UPLKINT` is generated and the status bit `UPLKFFULLERR` is set. It is up to the flight software to reset the uplink controller, aligning it to a first-packet-word boundary.

The BEP's Command Handler

A dedicated BEP "Command Task" thread is responsible for reading commands from the FIFO. Its *handle-Command* method is invoked from the BEP interrupt handler when the DPA hardware writes to the FIFO.

```
void CmdManager::handleCommand()
{
     swHousekeeper.report(SWSTAT_CMDMAN_HANDLED, 0);
     CmdPkt cmdPkt;        // Packet Instance
     // ---- Get packet buffer address to write to  ----
     unsigned short* buf = cmdPkt.getBufferAddress();
     // ---- Get the packet length ----
     cmdDevice.readFifo (&buf[0], 1);
     // ---- Check for bad command length, and recover if corrupt ----
     if ((buf[0] < 3) || (buf[0] > 256)) {
          swHousekeeper.report(SWSTAT_CMDMAN_BADLENGTH, buf[0]);
          handleError();
          return;
     }
     // ---- Read remainder of command into buffer ----
     cmdDevice.readFifo (&buf[1], buf[0] - 1);
     // ---- Log reception ----
     cmdEcho.openEntry (&cmdPkt);
     // ---- If command unuseable, treat as FIFO error ----
     if (cmdPkt.isValid() == BoolFalse) {
          swHousekeeper.report(SWSTAT_CMDMAN_INVALID, buf[0]);
          cmdEcho.closeEntry(&cmdPkt, CMDRESULT_INVALID_PKT);
          handleError();
          return;
     }
     // ---- Dispatch the command ----
     unsigned opcode = cmdPkt.getOpcode();
     CmdResult disposition = CMDRESULT_NO_HANDLER;
     if (handlerTable[opcode] != 0)
          disposition = handlerTable[opcode]->processCmd(&cmdPkt);
     // ---- Log disposition ----
     cmdEcho.closeEntry (&cmdPkt, disposition);
     // ---- Decrement pending packet count ----
     { IntrGuard guard; pktPending--; }
     // ---- If another, process the pending packet ----
     if (pktPending != 0)
          notify(EV_CMDINT);
}
```

The calls to *cmdDevice.readFifo(address, count)* do just what the name says—they read *count* 16-bit words from the FIFO into the named *address*. If there are fewer words ready to be read from the FIFO, the routine blocks until they become available. Since *readFifo()* runs with interrupts enabled, a truncated command will cause the BEP task managed to go on to other execution threads, returning to the command manager at 100 millisecond intervals until the whole command has arrived.

Flushing the FIFO

Since the DPA accepts no explicit external command to flush the FIFO, the only ways to resume control are to reboot the BEP or to continue sending 16-bit words until the number received matches the command length. If we choose the latter course, we must design a command that is sufficiently “legal” to be accepted by the ground software and the DPA hardware, but sufficiently “illegal” that it causes *handleCommand()* to take one of the calls to *handleError()* that will flush the command FIFO and report the error to software housekeeping:

```
void CmdManager::handleError()
{
     swHousekeeper.report(SWSTAT_CMDMAN_ERRCALLED, 0);
     // ---- Keep trying until we get no errors and an empty FIFO ----
     do {
           swHousekeeper.report(SWSTAT_CMDMAN_ERRRETRY, 0);
           // --- Disable receiver and reset the fifo and zero pending ---
           cmdDevice.disableReceiver();
           cmdDevice.resetReceiver();
           pktPending = 0;
           // --- Ensure no command words for minimum time ---
           sleep (pktDelay);
           // --- Reenable receiver prior to Fifo check ---
           cmdDevice.enableReceiver();
     } while ((cmdDevice.getErrStatus() != CmdDevNoErr) ||
           (cmdDevice.isFifoEmpty() != BoolTrue));
}
```

Moreover, it must do the job irrespective of the number of words needed to complete the command. The special command should have the following properties:

1. It must be at least 255 16-bit words in length in order to flush out a truncated command, which can be from 3 through 256 words long.
2. Beyond its first word, *i.e.,* its length, which must be 255 or 256, the value of its remaining words should be illegal as a length and illegal as a command type.
3. This restricts the value to range 0 or 41–256, excluding 192, 195, 204, and 240.
4. For simplicity, I have chosen a length of 256, followed by 255 zeroes.

It has been pointed out, that a long stream of zero bits might cause the uplink receivers to lose phase-key lock. This would be true if the command were uplinked as a single string, e.g., to reside in an SCS slot prior to executing, but our special packet is intended to be used only as a realtime command, where each 16 zero bits will be interspersed by 12 address bits which are guaranteed non-zero, providing the receivers with an ample supply of “edges” to maintain phase-key lock.

Testing

The task of verifying that the FIFO can be reliably flushed poses several problems for our ACIS engineering unit (EU) hardware simulator. It is usually commanded through an L-RCTU interface unit, by a UNIX workstation running a software script written in the *expect* dialect of the *tcl* language. However, (1) our software expects that each command begins with a valid length field, making it impossible to send a truncated command! (2) we have not needed to receive the DPA bilevel values before now, and (3) our *pmon* telemetry display is not configured to display EU bilevels. Here is what was done to circumvent these limitations:

1. Normal full-length commands are sent in the usual manner through a UNIX pipe that concatenates the *bcmd*, *sendCmds*, and *cclient* commands. *bcmd* translates each ASCII command into a 4-byte header followed by an array of 16-bit words; *sendCmds* translates this, adding further addressing information, and writing 24-bit words to *cclient*, so we must truncate *sendCmds* output by multiples of 3 bytes in order to send a

truncated command to the EU. In the *expect* script, we do this by killing and restarting the command pipe, inserting a *dd* command between *sendCmds* and *cclient*. The special *fifoflush* command packet cannot be created by *bcmd*; instead, we use a simple Perl command: "`perl -e 'print pack("v258",2,2,256)'`", which is passed directly into *sendCmds*.

2. The output from the EU runs from the L-RCTU into the *ltp2mnf* command which packages the telemetry into 1029-byte records that simulate Chandra minor frames. From *ltp2mnf*, the telemetry passes through *getPackets*, then *filterServer*, *FilterClient*, and *psci*, which lists the contents of packet headers. These are read by the *expect* script, to check that the commands have been received—or not—and that the script can move on, or fail.

3. lpt2mnf inserts the DPA bilevel flags into byte offset 137 of minor frame number 107. This is not the location used by flight telemetry, so software that reads the EU telemetry must be told to find the bilevels in this location, by supplying a special value to the `$ACISTTMFILE` environment variable that points to a file *"eng.ttm"* that defines the location. To test the "FIFO empty" flag, the *expect* script invokes a separate pipe consisting of *filterClient*, *ltlm*, and *awk* to check whether the flag is in the expected state. If not, the test fails.

4. The telemetry display routine *pmon* also expects to find ACIS bilevels at the flight location, so it has been updated to accept a special flag, `–U`, to instruct it that the bilevels are in the special location.

```
#! /bin/sh

export ACISSERVER ACISPORT

case $1 in

cmd)   ( bcmd |\
         sendCmds |\
         cclient $ACISSERVER $ACISPORT
       ) >>$ACISSERVER.err 2>&1
       ;;

trunc) ( bcmd |\
         sendCmds |\
     (     dd bs=3 count=$2 2>/dev/null ) |\
         cclient $ACISSERVER $ACISPORT
       ) >>$ACISSERVER.err 2>&1
       ;;

flush) ( perl -e 'print pack("v258",2,2,256)' |\
         sendCmds |\
         cclient $ACISSERVER $ACISPORT
       ) >>$ACISSERVER.err 2>&1
       ;;

blv)   rc=`filterClient -h $ACISSERVER |\
       ltlm -p61 -v -F $PWD/eng.ttm |\
       awk '/ value .* XACISBLA/ && sw++ > 0 { print int($3/128) ; exit }'`
       ps xww|awk '/ ltlm -p61 / { printf "kill %d %d\n", $1, $1-1 }' | sh
       exec [ "$rc" -eq $2 ]
       ;;

tlm)   filterClient -h $ACISSERVER | psci -m -u
       ;;

*)     echo "$0: bad argument: $1" >& 2
       exit 1
       ;;
esac

exit 0
```

The multi-purpose *"spawns.sh"* script (above) implements the various interfaces between the *"fifoflush.tcl"* script (below) and the L-RCTU. Called with the `CMD` argument, it sends one or more full-length commands to the EU; called with `TRUNC` and an integer, it sends no more than that number of 16-bit words to the EU; called with `FLUSH`, it sends the special *fifoflush* packet; called with `BLV`, it examines engineering packets from the EU and tests whether the `1STAT7ST` flag in the second major frame has the expected value; finally, called with `TLM`, it runs EU telemetry through *psci* to report the contents of packet headers and user pseudo-packets.

With this script to handle the interfaces, the *"fifoflush.tcl"* script itself is quite straightforward. Some procedures and other definitions have been omitted for greater clarity...

```
#! /usr/bin/expect
# ---- addPatch contents taken from the “compressall” patch ----
set patch "0x800e3460 {\n 0x27bdffe8 ... 0x8c42005c \n}"
set patchid 1000  # unused patch ID
# ---- Embed the EU test procedure library ----
source [lindex $argv 0]
# ---- Start an input command pipe ----
spawn spawns.sh cmd
set cmd_id $spawn_id
# ---- Start a telemetry output pipe ----
spawn spawns.sh tlm
sleep 1
# ---- Remove any existing patch with this id ----
send -i $cmd_id "remove $patchid patch {\n $patchid \n}\n"
command_echo . 22 "remove patch"
# ---- Send a full-length patch and remove it again
send -i $cmd_id "add $patchid patch $patchid $patch\n"
command_echo 1 21 "add patch $patchid"
send -i $cmd_id "remove $patchid patch {\n $patchid \n}\n"
command_echo 1 22 "remove patch"
# ---- Test various truncation lengths
foreach len { 1 2 3 4 100 200 250 251 252 253 254 255 } {
     # ---- Send a truncated addPatch and wait for “FIFO not empty”
     server trunc $len
     send -i $cmd_id "add $patchid patch $patchid $patch\n"
     wait_timeout 21 "addPatch $patchid"
     if {[catch {system spawns.sh blv 1} e]} {fail “FIFO empty\n$e”}
     puts “1STAT7ST: FIFO not empty”
     # ---- Send fifoflush packet, wait for bilevel flag to reset
     server flush 0
     if {[catch {system spawns.sh blv 0} e]} {fail “FIFO not empty\n$e”}
     puts “1STAT7ST: FIFO empty”
     # ---- Remove the patch (if it was ever added)
     server cmd 0
     send -i $cmd_id "remove $patchid patch {\n $patchid \n}\n"
     command_echo . 22 "remove patch"
     # ---- Send the addPatch again and then remove it
     send -i $cmd_id "add $patchid patch $patchid $patch\n"
     command_echo 1 21 "add patch $patchid"
     send -i $cmd_id "remove $patchid patch {\n $patchid \n}\n"
     command_echo 1 22 "remove patch"
}
# Done
pass "Done"
```

The *"fifoflush.tcl"* script runs automatically. After setting up sub-processes to pass commands to the EU and receive telemetry back, it issues a full-length *removePatches* command, followed by an *addPatch* and a *removePatches*, all relating to *patchId* 1000, waiting each time for the EU to echo a response. It ignores the *result* code from the first *removePatches*, since the script doesn't know whether that a patch of that index remained in the EU after a prior test. Then, in its `foreach` loop, it sends an *addPatch* command truncated to a chosen length. When no *commandEcho* is received back, it waits for engineering telemetry and checks that the `1STAT7ST` (FIFO empty) flag is set, *i.e.*, the FIFO is not empty. Still within the loop, it sends the special 512-byte *fifoflush* command packet, checks that the `1STAT7ST` flag is clear again, and sends full-length *removePatches*, *addPatch*, and *removePatches*, verifying that all three commands were received and executed correctly by the EU, *i.e., result*=1 in their *commandEcho* packets.

Glossary of *tcl* and *expect* commands

Here's a short explanation of those *tcl* and *expect* commands that are used in the *"fifoflush.tcl"* script. Note that the *tcl* language distinguishes between a variable's name and its value: the value is written with a `"$"` prefix, *e.g.*, `"puts $var"`, but is omitted otherwise, *e.g.*, `"set var 1"`, `"set var $val"`, etc.

`command_echo` *rc ctype text*	* Wait for the EU to echo a command of type *ctype* that returns a *result* code of *rc* in its *commandEcho* packet. If *rc* is specified as ".", ignore the value of the *result* code. If a 10-second wait times out, or if the *commandEcho* contains an unexpected *result* code, halt *"fifoflush.tcl"* and include *text* in the error message.
`catch` *cmd var*	Execute *cmd*. If it fails, return a non-zero value, and an error message in *var*.
`fail` *text*	* Terminate *"fifoflush.tcl"* with a failing grade, including *text* in the condemnatory message.
`foreach` *var arr cmds*	Execute one or more *tcl* commands, *cmds*, with *var* set in turn to each element of *array*.
`global` *var* [*var* ...]	Recognize *var* as the same variable across multiple procedures.
`if` *test cmds*	If *test* is non-zero, execute *cmds*.
`pass` *text*	* Terminate *"fifoflush.tcl"* with a passing grade, including *text* in the congratulatory message.
`puts` *text*	Write *text* to the standard output.
`send` [`-i` *id*] *text*	Send *text* to the command pipe with a spawn descriptor of *id*. In *"fifoflush.tcl"*, *text* must consist of an ASCII command in *bcmd* format.
`server` *mode arg*	* Close the previous command pipe and start *"spawns.sh"* to open a new one, passing arguments *mode* and *arg*.
`set` *var value*	Assign *value* to the variable named *var*.
`sleep` *seconds*	Do nothing for the specified number of *seconds*.
`source` *file*	Insert *tcl* commands and procedures from *file*.
`spawn` *script arg* [*arg* ...]	Execute *script* as an *expect* sub-task, passing arguments and returning a spawn descriptor in the variable `spawn_id`.
`system` *cmd* [*arg* ...]	Execute *cmd*, with optional arguments, in a UNIX command shell.
`wait_timeout` *ctype text*	* Wait for the EU to echo a command of type *ctype*, but expect the wait to timeout (*cf.* `command_echo`, above). If a legitimate *commandEcho* packet is received, halt *"fifoflush.tcl"* and include *text* in an error message.

* Local command defined in *fifoflush.tcl* or in the ACIS test library.

```
--------------------------------------------------------------------------------
Test: fifoflush
--------------------------------------------------------------------------------
Starting test on Fri May 14 16:01:04 EDT 2010
--------------------------------------------------------------------------------
spawn spawns.sh cmd
spawn spawns.sh tlm
stdin: userPseudo[1,0] SWSTAT_CMDMAN_HANDLED: 0
commandEcho 1 arrival=132 result=4 commandIdentifier=1000 commandOpcode=22
stdin: userPseudo[4,1] SWSTAT_CMDMAN_HANDLED: 0
commandEcho 2 arrival=145 result=1 commandIdentifier=1000 commandOpcode=21
stdin: userPseudo[6,2] SWSTAT_CMDMAN_HANDLED: 0
commandEcho 3 arrival=157 result=1 commandIdentifier=1000 commandOpcode=22
spawn spawns.sh trunc 1
addPatch 1000 timed out
1STAT7ST: FIFO not empty
spawn spawns.sh flush 0
1STAT7ST: FIFO empty
spawn spawns.sh cmd 0
swHousekeeping 4 startingBepTickCounter=0 endingBepTickCounter=642
stdin: userPseudo[36,3] SWSTAT_VERSION: ffff
stdin: userPseudo[53,4] SWSTAT_CMDMAN_HANDLED: 0
stdin: unrecognized commandOpcode 0 in commandEcho[54,3]
commandEcho 5 arrival=933 result=2 commandIdentifier=256 commandOpcode=0
stdin: userPseudo[55,5] SWSTAT_CMDMAN_ERRCALLED: 0
stdin: userPseudo[56,6] SWSTAT_CMDMAN_ERRRETRY: 0
swHousekeeping 6 startingBepTickCounter=642 endingBepTickCounter=1283
stdin: userPseudo[77,7] SWSTAT_VERSION: ffff
stdin: userPseudo[95,8] SWSTAT_CMDMAN_HANDLED: 0
commandEcho 7 arrival=1607 result=4 commandIdentifier=1000 commandOpcode=22
stdin: userPseudo[98,9] SWSTAT_CMDMAN_HANDLED: 0
commandEcho 8 arrival=1619 result=1 commandIdentifier=1000 commandOpcode=21
stdin: userPseudo[100,10] SWSTAT_CMDMAN_HANDLED: 0
commandEcho 9 arrival=1629 result=1 commandIdentifier=1000 commandOpcode=22
spawn spawns.sh trunc 2
addPatch 1000 timed out
1STAT7ST: FIFO not empty
spawn spawns.sh flush 0
1STAT7ST: FIFO empty
spawn spawns.sh cmd 0
swHousekeeping 10 startingBepTickCounter=1283 endingBepTickCounter=1923
stdin: userPseudo[119,11] SWSTAT_VERSION: ffff
stdin: userPseudo[135,12] SWSTAT_CMDMAN_HANDLED: 0
stdin: userPseudo[136,13] SWSTAT_CMDMAN_INVALID: 100
stdin: unrecognized commandOpcode 256 in commandEcho[137,7]
commandEcho 11 arrival=2196 result=9 commandIdentifier=1000 commandOpcode=256
stdin: userPseudo[138,14] SWSTAT_CMDMAN_ERRCALLED: 0
stdin: userPseudo[139,15] SWSTAT_CMDMAN_ERRRETRY: 0
swHousekeeping 12 startingBepTickCounter=1923 endingBepTickCounter=2564
stdin: userPseudo[161,16] SWSTAT_VERSION: ffff
stdin: userPseudo[178,17] SWSTAT_CMDMAN_HANDLED: 0
commandEcho 13 arrival=2856 result=4 commandIdentifier=1000 commandOpcode=22
stdin: userPseudo[180,18] SWSTAT_CMDMAN_HANDLED: 0
commandEcho 14 arrival=2868 result=1 commandIdentifier=1000 commandOpcode=21
stdin: userPseudo[183,19] SWSTAT_CMDMAN_HANDLED: 0
commandEcho 15 arrival=2878 result=1 commandIdentifier=1000 commandOpcode=22
spawn spawns.sh trunc 3
addPatch 1000 timed out
1STAT7ST: FIFO not empty
spawn spawns.sh flush 0
1STAT7ST: FIFO empty
spawn spawns.sh cmd 0
swHousekeeping 16 startingBepTickCounter=2564 endingBepTickCounter=3204
stdin: userPseudo[203,20] SWSTAT_VERSION: ffff
stdin: userPseudo[218,21] SWSTAT_CMDMAN_HANDLED: 0
commandEcho 17 arrival=3446 result=1 commandIdentifier=1000 commandOpcode=21
```

```
stdin: userPseudo[220,22] SWSTAT_CMDMAN_ERRCALLED: 0
stdin: userPseudo[221,23] SWSTAT_CMDMAN_ERRRETRY: 0
swHousekeeping 18 startingBepTickCounter=3204 endingBepTickCounter=3845
stdin: userPseudo[244,24] SWSTAT_VERSION: ffff
stdin: userPseudo[260,25] SWSTAT_CMDMAN_HANDLED: 0
commandEcho 19 arrival=4120 result=4 commandIdentifier=1000 commandOpcode=22
stdin: userPseudo[263,26] SWSTAT_CMDMAN_HANDLED: 0
commandEcho 20 arrival=4133 result=1 commandIdentifier=1000 commandOpcode=21
stdin: userPseudo[265,27] SWSTAT_CMDMAN_HANDLED: 0
commandEcho 21 arrival=4142 result=1 commandIdentifier=1000 commandOpcode=22
spawn spawns.sh trunc 4
addPatch 1000 timed out
1STAT7ST: FIFO not empty
spawn spawns.sh flush 0
1STAT7ST: FIFO empty
spawn spawns.sh cmd 0
swHousekeeping 22 startingBepTickCounter=3845 endingBepTickCounter=4485
stdin: userPseudo[287,28] SWSTAT_VERSION: ffff
stdin: userPseudo[300,29] SWSTAT_CMDMAN_HANDLED: 0
commandEcho 23 arrival=4709 result=1 commandIdentifier=1000 commandOpcode=21
stdin: userPseudo[302,30] SWSTAT_CMDMAN_ERRCALLED: 0
stdin: userPseudo[303,31] SWSTAT_CMDMAN_ERRRETRY: 0
swHousekeeping 24 startingBepTickCounter=4485 endingBepTickCounter=5126
stdin: userPseudo[327,32] SWSTAT_VERSION: ffff
stdin: userPseudo[342,33] SWSTAT_CMDMAN_HANDLED: 0
commandEcho 25 arrival=5370 result=1 commandIdentifier=1000 commandOpcode=22
stdin: userPseudo[344,34] SWSTAT_CMDMAN_HANDLED: 0
commandEcho 26 arrival=5385 result=1 commandIdentifier=1000 commandOpcode=21
stdin: userPseudo[347,35] SWSTAT_CMDMAN_HANDLED: 0
commandEcho 27 arrival=5397 result=1 commandIdentifier=1000 commandOpcode=22
spawn spawns.sh trunc 100
addPatch 1000 timed out
1STAT7ST: FIFO not empty
spawn spawns.sh flush 0
1STAT7ST: FIFO empty
spawn spawns.sh cmd 0
swHousekeeping 28 startingBepTickCounter=5126 endingBepTickCounter=5766
stdin: userPseudo[370,36] SWSTAT_VERSION: ffff
stdin: userPseudo[382,37] SWSTAT_CMDMAN_HANDLED: 0
commandEcho 29 arrival=5957 result=1 commandIdentifier=1000 commandOpcode=21
stdin: userPseudo[384,38] SWSTAT_CMDMAN_ERRCALLED: 0
stdin: userPseudo[385,39] SWSTAT_CMDMAN_ERRRETRY: 0
swHousekeeping 30 startingBepTickCounter=5766 endingBepTickCounter=6406
stdin: userPseudo[410,40] SWSTAT_VERSION: ffff
stdin: userPseudo[424,41] SWSTAT_CMDMAN_HANDLED: 0
commandEcho 31 arrival=6635 result=1 commandIdentifier=1000 commandOpcode=22
stdin: userPseudo[427,42] SWSTAT_CMDMAN_HANDLED: 0
commandEcho 32 arrival=6650 result=1 commandIdentifier=1000 commandOpcode=21
stdin: userPseudo[429,43] SWSTAT_CMDMAN_HANDLED: 0
commandEcho 33 arrival=6659 result=1 commandIdentifier=1000 commandOpcode=22
spawn spawns.sh trunc 200
addPatch 1000 timed out
1STAT7ST: FIFO not empty
spawn spawns.sh flush 0
1STAT7ST: FIFO empty
spawn spawns.sh cmd 0
swHousekeeping 34 startingBepTickCounter=6406 endingBepTickCounter=7046
stdin: userPseudo[453,44] SWSTAT_VERSION: ffff
stdin: userPseudo[464,45] SWSTAT_CMDMAN_HANDLED: 0
commandEcho 35 arrival=7222 result=1 commandIdentifier=1000 commandOpcode=21
stdin: userPseudo[466,46] SWSTAT_CMDMAN_ERRCALLED: 0
stdin: userPseudo[467,47] SWSTAT_CMDMAN_ERRRETRY: 0
swHousekeeping 36 startingBepTickCounter=7046 endingBepTickCounter=7686
stdin: userPseudo[494,48] SWSTAT_VERSION: ffff
stdin: userPseudo[506,49] SWSTAT_CMDMAN_HANDLED: 0
commandEcho 37 arrival=7883 result=1 commandIdentifier=1000 commandOpcode=22
```

```
stdin: userPseudo[508,50] SWSTAT_CMDMAN_HANDLED: 0
commandEcho 38 arrival=7898 result=1 commandIdentifier=1000 commandOpcode=21
stdin: userPseudo[511,51] SWSTAT_CMDMAN_HANDLED: 0
commandEcho 39 arrival=7908 result=1 commandIdentifier=1000 commandOpcode=22
spawn spawns.sh trunc 250
addPatch 1000 timed out
1STAT7ST: FIFO not empty
spawn spawns.sh flush 0
1STAT7ST: FIFO empty
spawn spawns.sh cmd 0
swHousekeeping 40 startingBepTickCounter=7686 endingBepTickCounter=8327
stdin: userPseudo[536,52] SWSTAT_VERSION: ffff
stdin: userPseudo[546,53] SWSTAT_CMDMAN_HANDLED: 0
commandEcho 41 arrival=8470 result=1 commandIdentifier=1000 commandOpcode=21
stdin: userPseudo[548,54] SWSTAT_CMDMAN_ERRCALLED: 0
stdin: userPseudo[549,55] SWSTAT_CMDMAN_ERRRETRY: 0
swHousekeeping 42 startingBepTickCounter=8327 endingBepTickCounter=8968
stdin: userPseudo[577,56] SWSTAT_VERSION: ffff
stdin: userPseudo[588,57] SWSTAT_CMDMAN_HANDLED: 0
commandEcho 43 arrival=9147 result=1 commandIdentifier=1000 commandOpcode=22
stdin: userPseudo[591,58] SWSTAT_CMDMAN_HANDLED: 0
commandEcho 44 arrival=9162 result=1 commandIdentifier=1000 commandOpcode=21
stdin: userPseudo[593,59] SWSTAT_CMDMAN_HANDLED: 0
commandEcho 45 arrival=9171 result=1 commandIdentifier=1000 commandOpcode=22
spawn spawns.sh trunc 251
addPatch 1000 timed out
1STAT7ST: FIFO not empty
spawn spawns.sh flush 0
1STAT7ST: FIFO empty
spawn spawns.sh cmd 0
swHousekeeping 46 startingBepTickCounter=8968 endingBepTickCounter=9608
stdin: userPseudo[620,60] SWSTAT_VERSION: ffff
stdin: userPseudo[628,61] SWSTAT_CMDMAN_HANDLED: 0
commandEcho 47 arrival=9734 result=1 commandIdentifier=1000 commandOpcode=21
stdin: userPseudo[630,62] SWSTAT_CMDMAN_ERRCALLED: 0
stdin: userPseudo[631,63] SWSTAT_CMDMAN_ERRRETRY: 0
swHousekeeping 48 startingBepTickCounter=9608 endingBepTickCounter=10249
stdin: userPseudo[660,64] SWSTAT_VERSION: ffff
stdin: userPseudo[670,65] SWSTAT_CMDMAN_HANDLED: 0
commandEcho 49 arrival=10396 result=1 commandIdentifier=1000 commandOpcode=22
stdin: userPseudo[672,66] SWSTAT_CMDMAN_HANDLED: 0
commandEcho 50 arrival=10409 result=1 commandIdentifier=1000 commandOpcode=21
stdin: userPseudo[675,67] SWSTAT_CMDMAN_HANDLED: 0
commandEcho 51 arrival=10421 result=1 commandIdentifier=1000 commandOpcode=22
spawn spawns.sh trunc 252
addPatch 1000 timed out
1STAT7ST: FIFO not empty
spawn spawns.sh flush 0
1STAT7ST: FIFO empty
spawn spawns.sh cmd 0
swHousekeeping 52 startingBepTickCounter=10249 endingBepTickCounter=10889
stdin: userPseudo[703,68] SWSTAT_VERSION: ffff
stdin: userPseudo[710,69] SWSTAT_CMDMAN_HANDLED: 0
commandEcho 53 arrival=10983 result=1 commandIdentifier=1000 commandOpcode=21
stdin: userPseudo[712,70] SWSTAT_CMDMAN_ERRCALLED: 0
stdin: userPseudo[713,71] SWSTAT_CMDMAN_ERRRETRY: 0
swHousekeeping 54 startingBepTickCounter=10889 endingBepTickCounter=11530
stdin: userPseudo[743,72] SWSTAT_VERSION: ffff
stdin: userPseudo[752,73] SWSTAT_CMDMAN_HANDLED: 0
commandEcho 55 arrival=11660 result=1 commandIdentifier=1000 commandOpcode=22
stdin: userPseudo[755,74] SWSTAT_CMDMAN_HANDLED: 0
commandEcho 56 arrival=11672 result=1 commandIdentifier=1000 commandOpcode=21
stdin: userPseudo[757,75] SWSTAT_CMDMAN_HANDLED: 0
commandEcho 57 arrival=11682 result=1 commandIdentifier=1000 commandOpcode=22
spawn spawns.sh trunc 253
addPatch 1000 timed out
```

```
1STAT7ST: FIFO not empty
spawn spawns.sh flush 0
1STAT7ST: FIFO empty
spawn spawns.sh cmd 0
swHousekeeping 58 startingBepTickCounter=11530 endingBepTickCounter=12170
stdin: userPseudo[786,76] SWSTAT_VERSION: ffff
stdin: userPseudo[792,77] SWSTAT_CMDMAN_HANDLED: 0
commandEcho 59 arrival=12247 result=1 commandIdentifier=1000 commandOpcode=21
stdin: userPseudo[794,78] SWSTAT_CMDMAN_ERRCALLED: 0
stdin: userPseudo[795,79] SWSTAT_CMDMAN_ERRRETRY: 0
swHousekeeping 60 startingBepTickCounter=12170 endingBepTickCounter=12811
stdin: userPseudo[827,80] SWSTAT_VERSION: ffff
stdin: userPseudo[834,81] SWSTAT_CMDMAN_HANDLED: 0
commandEcho 61 arrival=12909 result=1 commandIdentifier=1000 commandOpcode=22
stdin: userPseudo[836,82] SWSTAT_CMDMAN_HANDLED: 0
commandEcho 62 arrival=12924 result=1 commandIdentifier=1000 commandOpcode=21
stdin: userPseudo[839,83] SWSTAT_CMDMAN_HANDLED: 0
commandEcho 63 arrival=12936 result=1 commandIdentifier=1000 commandOpcode=22
spawn spawns.sh trunc 254
addPatch 1000 timed out
1STAT7ST: FIFO not empty
spawn spawns.sh flush 0
1STAT7ST: FIFO empty
spawn spawns.sh cmd 0
swHousekeeping 64 startingBepTickCounter=12811 endingBepTickCounter=13451
stdin: userPseudo[869,84] SWSTAT_VERSION: ffff
stdin: userPseudo[874,85] SWSTAT_CMDMAN_HANDLED: 0
commandEcho 65 arrival=13496 result=1 commandIdentifier=1000 commandOpcode=21
stdin: userPseudo[876,86] SWSTAT_CMDMAN_ERRCALLED: 0
stdin: userPseudo[877,87] SWSTAT_CMDMAN_ERRRETRY: 0
swHousekeeping 66 startingBepTickCounter=13451 endingBepTickCounter=14092
stdin: userPseudo[910,88] SWSTAT_VERSION: ffff
stdin: userPseudo[916,89] SWSTAT_CMDMAN_HANDLED: 0
commandEcho 67 arrival=14173 result=1 commandIdentifier=1000 commandOpcode=22
stdin: userPseudo[919,90] SWSTAT_CMDMAN_HANDLED: 0
commandEcho 68 arrival=14188 result=1 commandIdentifier=1000 commandOpcode=21
stdin: userPseudo[921,91] SWSTAT_CMDMAN_HANDLED: 0
commandEcho 69 arrival=14197 result=1 commandIdentifier=1000 commandOpcode=22
spawn spawns.sh trunc 255
addPatch 1000 timed out
1STAT7ST: FIFO not empty
spawn spawns.sh flush 0
1STAT7ST: FIFO empty
spawn spawns.sh cmd 0
swHousekeeping 70 startingBepTickCounter=14092 endingBepTickCounter=14732
stdin: userPseudo[954,92] SWSTAT_VERSION: ffff
stdin: userPseudo[956,93] SWSTAT_CMDMAN_HANDLED: 0
commandEcho 71 arrival=14759 result=1 commandIdentifier=1000 commandOpcode=21
stdin: userPseudo[958,94] SWSTAT_CMDMAN_ERRCALLED: 0
stdin: userPseudo[959,95] SWSTAT_CMDMAN_ERRRETRY: 0
swHousekeeping 72 startingBepTickCounter=14732 endingBepTickCounter=15373
stdin: userPseudo[994,96] SWSTAT_VERSION: ffff
stdin: userPseudo[998,97] SWSTAT_CMDMAN_HANDLED: 0
commandEcho 73 arrival=15422 result=1 commandIdentifier=1000 commandOpcode=22
stdin: userPseudo[1000,98] SWSTAT_CMDMAN_HANDLED: 0
commandEcho 74 arrival=15437 result=1 commandIdentifier=1000 commandOpcode=21
stdin: userPseudo[1003,99] SWSTAT_CMDMAN_HANDLED: 0
commandEcho 75 arrival=15449 result=1 commandIdentifier=1000 commandOpcode=22

===Done===

*** PASS ***
Ending test on Fri May 14 16:27:47 EDT 2010
--------------------------------------------------------------------------------
```