jive5ab

The Swiss Army knife of (e-)VLBI

Harro Verkouter
Archaeology

$> cd ~/src/jive5ab
$> grep 'int main(' *cc

**test.cc:**

```cpp
int main(int argc, char** argv) {
```
Archaeology

- aug 2007
  - 256Mbps e-VLBI w/ China demo’ed
  - severely hacked Haystack Mark5A software
  - StreamStor problems?
    - crashing, hanging, ...
XLRReadData

Syntax:

\[ \text{XLR\_RETURN\_CODE XLR\_ReadData( SHANDLE xDevice, ULONG Buffer, ULONG AddrHigh, ULONG AddrLow, ULONG XferLength );} \]

Description:

XLRReadData reads data from the StreamStor device. This function is identical to XLRRead without the structure to pass request parameters.

The address of the requested data must be an eight-byte-aligned value.

If the StreamStor is in bank mode, this command will read data from the currently selected bank.

Parameters:

- \text{xDevice} is the device handle returned from a previous call to XLROpen.
- \text{Buffer} is the address of the user memory buffer to hold the requested data.
- \text{AddrHigh} is the upper 32 bits of a 64-bit byte address of the requested data.
- \text{AddrLow} is the lower 32 bits of a 64-bit byte address of the requested data.
- \text{XferLength} is the number of bytes requested.

Return Value:

On success, this function returns XLR\_SUCCESS.
On failure, this function returns XLR\_FAIL.

Usage:

```c
SHANDLE xDevice;
XLR\_RETURN\_CODE XLR\_returnCode;
ULONG myBuffer[4096];

xDevice = XLROpen( 1, xDevice );

//AddrHigh and AddrLow must represent an appropriately aligned address.
xLR\_returnCode = XLR\_ReadData( xDevice, myBuffer, 0, 0x0F120000, sizeof(myBuffer) );
```

See Also:

XLRRead, XLRSetMode, XLRSetBitMode, XLRSetBankMode and XLRSelectBank.
test program ‘test.cc’

• read data from StreamStor ≤ 1 day
• implement proto “disk2net” ≤ 1 a week
• development slowed a bit... still ...

We may be on to something!
test program ‘test.cc’

- add remote command interface
  - internal command execution system still in use today
- implement VSI/S style commands; in 2012:
  - Mark5A command set v. 2.73
  - Mark5B command set v. 1.12
  - jive5ab specific
Supported platforms in 2012

• Mark5A, B and partially C
• Intel CPU based
  – GNU c++ compiler
  – POSIX compliant O/S
    • Linux
      – Debian, RedHat, Gentoo
    • Mac OSX
• 32 and 64 bit
  – all platforms
  – not on Mark5 if StreamStor support needed
Dataflow e-VLBI

Mark5 system

PCI bus

diskmodules

StreamstorCard

I/O board

formatter/correlator

internet

jive5ab
VLBI connectivity: *src2dst*

```
net
- file
- fill
- jive5ab

Mark5 system

- disk
- StreamstorCard
- I/O board

in/out
```
VLBI connectivity: \textit{src2dst}

- src
- dst

Network (net)

File (file)

Fill (fill)

Intel based PC

Disk (disk)

StreamstorCard

Inputboard

In/Out (in/out)
VLBI connectivity: $src2dsts \ (p.)$
VLBI connectivity: \( src2dsts \ (p.) \)

Mark5 system

- jive5ab
- StreamstorCard
- I/O board

net

real-time reception + recording @ correlator

disk

out
VLBI connectivity: $src2dsts\ (p.)$

Mark5 system

future e-VLBI: always recording, 2$^{nd}$ transfer on demand
VLBI connectivity:

- disk2net, disk2out, disk2file,
- in2net, in2disk, in2fork, in2file
- net2out, net2disk, net2file, net2check, net2sfxc,
- fill2net, fill2file, fill2out
- spill2net, spid2net, spin2net, spin2file, splet2net, splet2file
- spill2file, spid2file, spif2file, spif2net
- file2check, file2mem
- in2mem, in2memfork, mem2net
## Supported network protocols

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>udp</td>
<td><em>mtu</em> sized packets with 64 bit sequence number. Receiver can compensate reordering/loss (VTP)</td>
</tr>
<tr>
<td>pudp</td>
<td>Plain <em>mtu</em> sized udp packets, NO sequence numbers. Can be used on reliable (local) links</td>
</tr>
<tr>
<td>tcp</td>
<td>Standard tcp</td>
</tr>
<tr>
<td>rtcp</td>
<td>'Reverse' tcp, reverses <strong>connection</strong> direction between sender/receiver for firewalled receivers. Data direction unchanged.</td>
</tr>
<tr>
<td>unix</td>
<td>Unix socket on local machine</td>
</tr>
</tbody>
</table>
Built-in network statistics

- RFC4737 compliant (cvs head)
- “evlbi?”
  - packet loss
  - packet reordering
  - extent of reordering
- “evlbi = <formatstring>”
  - “%l” amount of packets lost, “%L” loss as %
  - “%u” unix timestamp, “%U” human readable
  - “%o” amount of out-of-order packets, “%O” as %
  - few more

For use with udp/VTP only!
Supported dataformats

- Mark4
- VLBA
- Mark5B
  - up to 4096Mbps!
- VDIF
  - including legacy
- none
  - allows anonymous transfer of data
    - set “mode = none”
Data processing - 2010/2011

- significantly improved data processing core
- programmatically build processing chains
  - decide at runtime whether to include steps
  - each step runs in own thread
  - data passed via ‘work’ queue
- build collection of simple steps
  - enormous re-use of steps - mix ‘n match to taste!
- allows easy
  - experimentation with variations
  - addition of completely new steps
Data processing - 2010/2011

SRC

network
formatter
file
...

proc

compression
filtering
dbg
...

proc

... 

DST

correlator
diskmodule
file
network
...

// The hardware has been configured, now start building the processing chain

chain  c;

if( toqueue ) {
    c.add(&fifo_queue_writer, 1, queue_writer_args(&rte));
} else {
    c.add(&fiforeader, 10, fiforeaderargs(&rte));
}

// If compression requested then insert that step now
if( rte.solution ) {
    DEBUG(0, "in2net: enabling compressor " << dataformat << endl);
    if( dataformat.valid() ) {
        c.add(&framer<frame>, 10, framerargs(dataformat, &rte));
        c.add(&framecompressor, 10, compressorargs(&rte));
    } else {
        c.add(&blockcompressor, 10, &rte);
    }
}

// Write to file or to network
if( tofile ) {
    c.add(&fdwriter<block>, &open_file, filename, &rte);
} else {
    c.add(&netwriter<block>, &net_client, networkargs(&rte));
}

} // set everything running

  c.run();
if( toqueue ) {
}
else {

    // If compression requested then insert that step now
    if( rte.solution ) {
        if( dataformat.valid() ) {

            } else {

            }
    } else {

    }

} // Write to file or to network
if( tofile ) {

} else {

}
chain c;

if( toqueue ) {
    c.add fifo_queue_writer
} else {
    c.add fiforeader
}

if( rte.solution ) {
    if( dataformat.valid() ) {
        c.add framer
        c.add framecompressor
    } else {
        c.add blockcompressor
    }
}

if( tofile ) {
    c.add fdwriter
} else {
    c.add netwriter
}
}

c.run();
Built-in performance statistics

• “tstat?”
  – human readable
    • time elapsed since last call
    • name of the thread
    • amount of bytes/s processed by each thread since then
  – global ‘time of last call’
    • problems if > 1 users polling

• “tstat=“
  – machine readable
    • unix timestamp
    • name of the thread
    • raw bytecounts for each thread
Feature: data compression

64 bits

data read

data written
Feature: data compression

- used to bring down output bandwidth
  - 1024Mbps observing > 1Gbps network link
- ‘channel dropping’
  - allows user to select which bitstream(s) can be dropped
  - unused bits will be overwritten by used bits
- just-in-time compression/decompression routine
  - code generated, compiled + linked by jive5ab using system gcc
  - dynamically loaded back into jive5ab
- use “trackmask=0x....” to enable
  - max 64 bit; default bit is ‘0’ if less than 64 bits specified
  - “trackmask = 0” disables the feature
Feature: debug + testing

• in2file
• file2check
  – print all timestamps in file
• net2check
  – print timestamp if integer second changes
• fill2[net|file|out]
  – generate frames as per set data format
    • syncword
    • timestamp
    • fixed or incrementing pattern (fixed = increment 0 ...)
  – honours compression “trackmask=“
Feature: cornerturning (2012)

input: one buffer with $N$ channels

output: $N$ buffers with one channel each
Feature: cornerturning (2012)

- **input**
  - packed data format, carrying $N$ channels
    - Mark4, VLBA, Mark5B, multi channel VDIF
  - from *any* input!

- **output**
  - single channel, multi thread VDIF
  - multi channel, multi thread VDIF
  - threads individually routable
    - different files OR different network destinations
    - including ‘nowhere’
    - not yet mixed
  - output VDIF framesize can be != input frame size
    - allows reframing in multiples of input channel size
Feature: cornerturning (2012)

• hand crafted Intel SSE2 SIMD assembler code
  – for specific formats
    • Mark4/VLBA 4/8 channel fanout-2 (512Mbps)
    • 8 channel Mark5B (512Mbps)
  – breaking data up
    • 4x8bit, 2x16bit, 4x16bit, 2x32bit
  – swapping Mark5B sign-mag bits
    • VDIF and Mark5B have opposite ‘endiannes’

• dynamic channel extractor
  – relatively slow, just-in-time code generation + loading
  – “32>[0, 3][13, 19]....”
    • from each 32 bits
      – extract bits 0, 3 into channel 0
      – extract bits 13, 19 into channel 1
  – accommodate any format
Feature: cornerturning (2012)

Splitting/cornerturning can be daisy-chained

- 1024Mbps Mk4 = 16 channel obs = 2 x 8 channel Mk4 layout
  - split into 2x32bit
  - use 8 channel Mk4 cornerturner on both parts
- 4096Mbps = 32 channel Mk5B obs = 4 x 8 channel Mk5B layout
  - split into 4x16bit
  - use 8 channel Mk5B cornerturner on each of the four parts
- the dynamic channel extractor can be part of it too
1024Mbps/file, net, disk

UniBoard

Mark5A/B

FlexBuff

Mark5C

subband 0

subband 1

subband 2
Bugs

- No documentation
- No documentation
- No documentation
- parameter space is huge
  - chance of combinations which will #FAIL
- there’s always verkouter@jive.nl ...
Thanks for your attention!
QuickTime™ and a decompressor are needed to see this picture.
Primary objective
Primary objective
Packet size + spacing for UDP

• “mtu = <number>” (bytes)
  – jive5ab computes max payload
    • depending on format and boundary conditions
    – resize internal buffers to integral #-of-packets
    – default is 1500

• “ipd = <number>” (microseconds)
  – packet spacing absolutely necessary for UDP
  – timing of next packet is wall-clock time based
    • not relative or ‘roughly that time’
QuickTime™ and a decompressor are needed to see this picture.

FiLa10G

Station

Mark5C

1x4096Mbps

harrobox

= jive5ab

JIVE

sfxc-0

sfxc-1

sfxc-2

sfxc-3
// The hardware has been configured, now start building the processing chain

chain c;

if( toqueue ) {
    c.add(&fifo_queue_writer, 1, queue_writer_args(&rte));
} else {
    c.add(&fifo_reader, 10, fiforeaderargs(&rte));
}

// If compression requested then insert that step now
if( rte.solution ) {
    DEBUG(0, "in2net: enabling compressor " << dataformat << endl);
    if( dataformat.valid() ) {
        c.add(&framer<frame>, 10, framerargs(dataformat, &rte));
        c.add(&framecompressor, 10, compressorargs(&rte));
    } else {
        c.add(&blockcompressor, 10, &rte);
    }
    // Write to file or to network
    if( tofile ) {
        c.add(&fdwriter<block>, &open_file, filename, &rte);
    } else {
        c.add(&netwriter<block>, &net_client, networkargs(&rte));
    }
} else {
    c.add(&blockcompressor, 10, &rte);
}

// set everything running

}