PF_ZIO: Using Network Frames to Convey I/O Data and Meta-Data

http://www.ohwr.org/projects/zio

Alessandro Rubini, Federico Vaga, Simone Nellaga
Independent consultants in Pavia, Italy.
Working for CERN "hardware and timing" group
Channels, Csets, Devices

ZIO is concerned with I/O channels
- A channel is a single input or output wire

Channels are grouped in "channel sets"
- All channels in a cset share a trigger instance
- All channels in a cset use the same buffer type

Csets are grouped into devices
- A device is the register/unregister atomic entity
- Several devices of the same type can coexist
The atomic data item in ZIO is a block

- A block hosts data samples
- It also hosts meta-data (control information)
- Data within ZIO never travels without meta-data
The Control

This area hosts attributes for the device and for the currently active trigger.

Device and trigger are each characterized by 16 "standard" attrs and 32 "extended" attrs. A bit-mask states which attrs are active.

Each attribute is a 32-bit word

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>V, v, A, a, sequence, nsamples, ssize, nbits</td>
</tr>
<tr>
<td>0x10</td>
<td>fam, type, host-identification, device-id</td>
</tr>
<tr>
<td>0x20</td>
<td>cset, chan, device name</td>
</tr>
<tr>
<td>0x30</td>
<td>tstamp: secs, tstamp: ticks</td>
</tr>
<tr>
<td>0x40</td>
<td>tstamp: bins, mem-addr, reserved</td>
</tr>
<tr>
<td>0x50</td>
<td>flags, trigger name</td>
</tr>
<tr>
<td>0x60</td>
<td>TLV record for optional extra information</td>
</tr>
</tbody>
</table>
ZIO Device types

ZIO supports both input and output since inception

Our device types are "analog", "digital" or "time"

Input block:
- Data collected at a specific time or event

Output block:
- Data to be emitted at a specific time or event

"Time" channels:
- Digital pulses from/to laboratory equipment
- (No data is associated to a time channel)
The Hard Requirements behind ZIO

Hardware timestamps (better than 1ns precision)

Big data blocks (stripes of many samples)

Off-line creation/gathering of data blocks

High data rate

Easy monitoring of a diverse I/O environment

Support for several (many) boards of the same type
Design Choices behind ZIO

Sysfs-based configuration

No ioctl(2) thank you

Centralized locks (drivers must ignore the issue)

Modular design (each object should be replaceable)

Documented and stable, with version control
All Items in a ZIO Framework

- The block is overall blue
- Control
- Cyan
- Data
- Darker

Colors:
- Luser
- Lemon
- Fops
- Forest
- Socket
- Salmon
- Buffer
- Brown
- Trigger
- Tomato
- Periph.
- Purple
- Network
- Neutral
ZIO pipeline, User to Hardware and Back

write ➔ alloc_block ➔ push_block ➔ raw_io
store_block ➔ retr_block ➔ data_done
free_block

read ➔ retr_block ➔ pull_block ➔ raw_io
free_block
alloc_block
store_block ➔ data_done
Each channel is exported to user space as two char devices

- You can use blocking-read or poll on control, then read data
- Some users can choose to ignore control and just read data
- Other users can read control and ignore undesired data
- The "current" control block is exported, read-only, in sysfs
- Input and Output are completely symmetric
Users can change the buffer at runtime

- If you don't need the timestamp for each and every block...
- You can save buffering memory preserving the data model
- This is not the default, but can be chosen through sysfs
This is a buffer using vmalloc instead of kmalloc

- The control includes an "mmap_offset" field
- You avoid one data copy with DMA-capable peripherals
Defining PF_ZIO for I/O Blocks

The ZIO metadata+data model reminds network frames

- There are some advantages in socket programming
- So we chose to implement PF_ZIO as a socket family
- The control already includes an addr_zio structure...

<table>
<thead>
<tr>
<th>0x00</th>
<th>V</th>
<th>v</th>
<th>A</th>
<th>a</th>
<th>sequence</th>
<th>nsamples</th>
<th>ssize</th>
<th>nbis</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x10</td>
<td>fam</td>
<td>type</td>
<td>host-identification</td>
<td>device-id</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x20</td>
<td>cset</td>
<td>chan</td>
<td>device name</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x30</td>
<td>timestamp: secs</td>
<td>timestamp: ticks</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x40</td>
<td>timestamp: bins</td>
<td>mem-addr</td>
<td>reserved</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x50</td>
<td>flags</td>
<td>trigger name</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x60</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

This area hosts attributes for the device and for the currently active trigger.

Device and trigger are each characterized by 16 "standard" attrs and 32 "extended" attrs. A bit-mask states which attrs are active.

Each attribute is a 32-bit word

| 0x1F0 | TLV record for optional extra information |
We map the three standard socket types to ZIO blocks

- The code is implemented as a ZIO buffer
- Triggers and Peripheral drivers are unaffected
The ZIO pipeline, with zio-buf-sock active
Communication Paths Within a Host
Communication Paths Across Hosts
Our frame format supports inter-host communication,

- The "zio" network interface is an Ethernet card
- We carry around an Ethernet header for each block
- sockaddr_zio already has "host type" and "host-id"
PF_ZIO is not (only) ZIO over Ethernet

The new PF_ZIO/AF_ZIO is not about Ethernet Frames
- The PF_ZIO address space is about I/O channels
- Frames are used to exchange I/O blocks
- Typically, the ZIO network lives inside a single host

Why using a "networkless" network protocol?
- A host may need to drive hundreds of channels
- Sockets prove better than many char devices
- Zero-copy networking helps with high data rates
- Sniffing is a boost during debugging

(ETH_P_ZIO is just a special case of the idea)
Implementation Status

device: zio-zero (input and output)
device: zio-loop (for stress-testing and diagnostics)
device: line discipline (input: UART or pty for stress-test)
device: GPIO (input and output)
device: AD7888/AD7887 (SPI ADC)
device: fmc-based TDC/DTC
device: fmc-fine-delay (input and output: 10ps resolution)
device: fmc-based 100MS ADC
  trigger: kernel timer
  trigger: high-resolution timer
  trigger: transparent trigger (user/device driven)
  trigger: external interrupt or external GPIO
buffer: "kmalloc"
buffer: "data" (SOCK_STREAM alike, coalescing blocks)
buffer: "vmalloc" (mmap-capable)
  sockets: SOCK_DGRAM and SOCK_RAW (sock STREAM almost working)
tools: zio-dump (control and data)
tools: zio-cat-file (demonstrating mmap for input channels)
tools: pfzio-send and pfzio-receive (like netcat)
Thank you for your attention

http://www.ohwr.org/projects/zio

git://ohwr.org/misc/zio.git

http://www.ohwr.org/projects/zio/documents
http://www.ohwr.org/projects/zio/wiki