nurdlib – new DAQs in a jiffy

“Copy, paste, hack, make, debug, rinse, repeat...”
No thx.

14.3.2017

Bastian Löher
Most slides courtesy H. T. Törnqvist

Introduction

- Text-configurable readout
  - YAAAAARlib (Yet Another Attempt At An Absolute Readout library)
  - nurdlib (NUstar ReaDout library)
- Reduce time and work to setup a VME crate
- Many tested features
- Easy debugging
- In the future, generated human-readable configs (based on cabling documentation!) and unpacker specification

The (good?) old way

1) 27 f_users scattered all over
2) Copy code
3) Hack in parameters from memory/manuals
4) Glue together
5) Build, go to 2 until successful
6) Run, go to 2 (or 3) until data are (seemingly) pretty
7) Do the same for each crate / computer
8) Cross your fingers (optional)

The nurdlib way

http://web-docs.gsi.de/~land/nurdlib/

1) Build nurdlib once

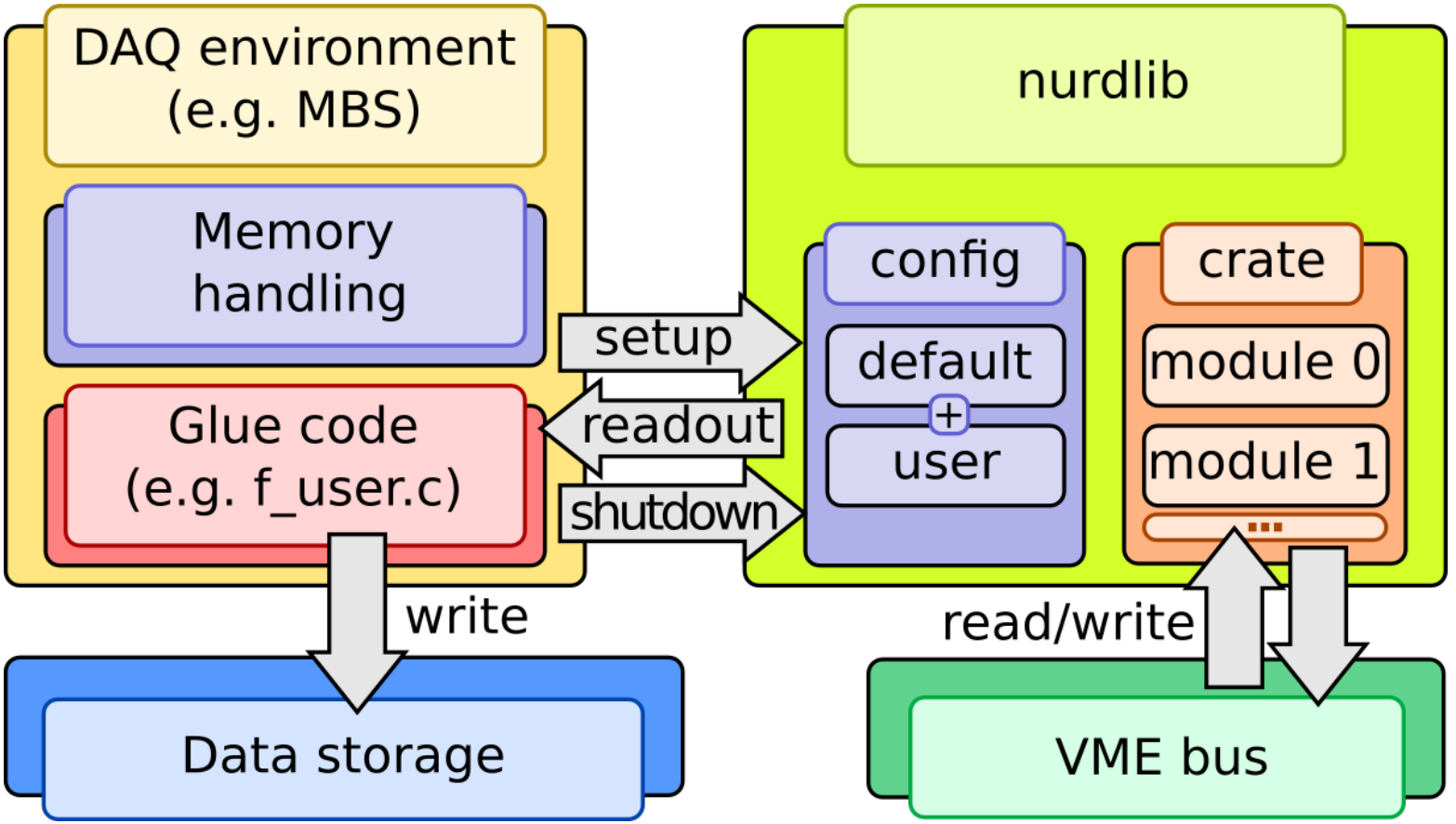
2) Write small config file for each crate

3) Lean back and smile (mandatory)

(Disclaimer: at least this is the plan...)

Overview

http://web-docs.gsi.de/~land/nurdlib/



Advantages with nurdlib

- Many (potentially complex) features supported
- Git version control from the start
  - Collaborative work, continuous integration testing
  - Bug history raids, bisecting, all the good stuff
- Useful features may stay
- nurdlib battered with unit tests
  - Config parser, getter, utility code, default module modes, crate state, and more...
  - Tests added after bugs to prevent regression
  - Performed offline
- Build system is GNU make + portability ensured by htools add-on
- Code is strictly C89, so all compilers should understand

Currently supported modules

- Caen
  - v775, v785, v792, v820, v830, v895, v965, v1190, v1290, (dt5790)
- GSI
  - sam, tacquila, tridi(trlo2), triva, vetar, vftx2, vulom(trlo2), vuprom(TDC)
- Mesytec
  - madc32, mqdc32, mtdc32
- PNPI - cros3
- Struck SIS - sis3316
- Dummy module for testing without hardware
- + Anything we can get our hands on

Other nice features

- Not bound to underlying DAQ software (e.g. MBS)
- Runs on RIO2+3+4, MVME, x86PC, raspberry Pi (arm)
- DMA block-transfer
  - Static and dynamic mapping automatic, if hardware supports it
  - Mapping ‘poked’ before execution to prevent system hanging
- Multi-events
  - nurdlib computes and configures designated TRLO2 trigger module
- Sane module default settings are provided
- Automatic pedestal estimation → automatic thresholds
- Adaptive CVT setting

Other nice features

- Online data-checking
  - Fast simple verification during readout
  - Thorough testing can be done when processor not busy
    - ‘Shadowed’ readout
- Per module log level setting (when debugging / setting up a single module)
- Module memory testing
- Runtime control program
  - List modules, dump modules, change register settings, get DAQ timestamp

But this must be huge!

- We are simple-minded, we like it simple
  - Simple code → simple states
  - Spaghetti belongs on a plate
  - Regression testing (don’t trust myself (or others...))
- 1.25 MB “online” source code
  - 197 kB general code + 40 kB default configs
  - 800 kB module code
    - Module support can be chosen when building
  - 157 kB test code + 3.5 kB test configs
  - 150 kB generated code
  - 40 kB control program

So what does a config file look like?

Caen and GSI TDC with multi-events and master TRIDI

```
log_level = info

# The second parameter is an ID for the crate.
CRATE("MyCrate", 0) {
        multi_event = true
        GSI_TRIDI(0x02000000) {
                trlo2_master = true
        }
        CAEN_V775(0x00400000) {
                time_range = 600 ns
        }
        GSI_VFTX2(0x09000000) {
        }
}
```

DAQ broke? git diff!

Copy-paste from terminal “`git diff --word-diff`”

```
diff --git a/main.cfg b/main.cfg
index f345ad7..a6462b9 100644
--- a/main.cfg
+++ b/main.cfg
@@ -7,8 +7,10 @@ CRATE(“MyCrate”, 0) {
                trlo2_master = true
        }
        CAEN_V775(0x00400000) {
                time_range = [-600-]{+300+} ns
        }
        GSI_VFTX2(0x09000000) {
                {+channel_enable = 0..5+}
{+          clock_input = external+}
        }
}
```

“Hmm, too short TDC window, or missing clock signal?”

Default config example

Copy-paste from current VFTX2 default config

```
channel_enable = 0..31
channel_invert = ()

# Don’t print header if no channels fired
verbose = false

# lemo or ecl.
trigger_input = lemo

# internal or external, internal by default because
# external without a clock requires a module reboot.
clock_input = internal

GATE {
        time_after_trigger = -1us
        width = 1us
}
```

Integration with DAQ

Simple interface to underlying DAQ code

```
# Setup nurdlib (e.g. at f_user_init())
struct Crate *nurdlib_setup(config_path)

# Shutdown nurdlib (e.g. atexit())
nurdlib_shutdown(crate)

# In the readout loop (e.g. f_user_readout())
crate_readout_prepare(crate) // before any readout
crate_readout(crate, buffer, size) // actual readout
crate_readout_finalize(crate) // after readout

# For trigger logic data
crate_tpat_get()
crate_timestamp_get()
crate_master_get()
```

Integration with DAQ

- Example glue code for R3B experiments exists
  - mbs ↔ r3bfuser ↔ nurdlib
- You probably need your own special tricks
- r3bfuser could be a basis
  - handles scalers from trlo and sends them via udp
  - reads timestamps and tpat from trigger logic
  - pileup histogramming
  - onspill/offspill trigger handling

Full DAQ - MBS

- What does a functioning DAQ directory look like?
  - htools (portability + test facility)
  - nurdlib (readout library)
  - trloii (trigger logic library)
  - r3bfuser (glue code)
  - mbs (DAQ, not actually inside the DAQ directory)
  - r4l-10 (configuration for a single node)
    - main.cfg (nurdlib config)
    - setup_standalone.usf (DAQ config)
    - tridi.trlo (trigger logic config)
    - trloii_setup.sh (triva mimic setup)
  - r4l-xx ...

Full DAQ - MBS

- What does a functioning DAQ directory look like?
  - htools (portability + test facility)
  - nurdlib (readout library)
  - trloii (trigger logic library)
  - r3bfuser (glue code)
  - mbs (DAQ, not actually inside the DAQ directory)
  - r4l-10 (configuration for a single node)
    - main.cfg (nurdlib config)
    - setup_standalone.usf (DAQ config)
    - tridi.trlo (trigger logic config)
    - trloii_setup.sh (triva mimic setup)
  - r4l-xx ...

These are static! No changes needed!

Full DAQ - drasi

- What does a functioning DAQ directory look like?
  - htools (portability + test facility)
  - nurdlib (readout library)
  - trloii (trigger logic library)
  - r3bfuser (glue code)
  - drasi (DAQ)
  - r4l-11 (drasi)
    - main.cfg (nurdlib config)
    - master.sh (DAQ config)
    - tridi.trlo (trigger logic config)
    - trloii_setup.sh (triva mimic config)
  - x86l-31 (event builder)
    - eb.sh (event builder config)

Add-Ons

- nurdlib does not need, but can make use of
  - TrLoII trigger logic on VULOM or TRIDI
  - TRIVA style trigger logic
  - TRIXOR/PEXOR in development for tamex/febex/nxyter

Timeguide (when is it ready?)

- Started by A. Charpy, Chalmers
- Continued by H. Törnqvist + B. Löher, TUD and M. Munch, Århus

- Used in every R3B experiment since 2014
  - Initially not for all detectors/systems, but now nearing completion!
- Used in some places at ESR, FRS
- External users: Riken, Duke University, Århus, ISOLDE@CERN
- Under active development
  - Everyone is welcome to try and test and submit bug reports!
- Nurdlib will be central ingredient of Nustar DAQ

References

- nurdlib: http://web-docs.gsi.de/~land/nurdlib/
- trloii: http://fy.chalmers.se/~f96hajo/trloii/
- htools: https://hanstt@bitbucket.org/hanstt/htools
- r3bfuser: /u/htoernqv/repos/r3bfuser.git
- ucesb: http://fy.chalmers.se/~f96hajo/ucesb/
- installdaq: /u/bloeher/git-bare/installdaq.git

Requests are welcome

- Would you like something?
- Are you worried about a feature?
- Are you even a willing tester?