Data Acquisition for Nuclear Physics Experiments

Summer Science Series – July 19, 2022
Virtual Presentation

David Abbott - FEDAQ Group
Jefferson Lab – Physics Division
Data Acquisition

• **data acquisition**

  Data acquisition is defined as the process of collecting and organizing information.

• **Anatomy of this DAQ Talk:**
  • Focus on DAQ for nuclear physics experiments.
  • What are all the pieces of a typical DAQ system?
  • How we have implemented them here at Jefferson Lab?
  • What are we working on for the future?
  • Hopefully I can make it interesting...

**DAQ support at Jefferson Lab**

FEDAQ (Fast Electronics and Data Acquisition)
EPSCI (Experimental Physics Scientific Computing Infrastructure)

We can sometimes be found here on the 2nd Floor of F-Wing in CEBAF Center
Detection and data acquisition

• First nuclear physics experiments (early 1900s) in Manchester, UK by Earnest Rutherford, Hans Geiger and Ernst Marsden.
  – **Gold foil experiments** – confirmed the existence of the positively charged nucleus.
  – Recorded the positions of light flashes on a fluorescent screen generated by alpha particles passing through and being scattered by the foil.

In General:

• When a particle interacts with a target, resultant particles from the interaction enter a detector.
• Various sub-detectors are designed to measure properties of these particles (**type**, **energy**, **trajectory**) initially as electrical signals.
• Three basic types of measurements are - **charge**, **time** and **count**.
• Electronics and software convert electrical signals into digital data that can be stored and later analyzed.
  
  ADC = charge TDC = time Scaler = count
• All of the data generated from one “interaction” we call an **Event**.
The Event as a Unit of Data

• We typically assume data from one event has no history. It doesn’t depend upon events that went before and doesn’t influence later events.

• Events occur with random timing.
  – Average rates run from a few Hz to 100+ kHz
  – Hardware may not be ready for new data.
    – **System dead time**.
  – Events may overlap in time - **Event pileup**.
  – Peak event rates can be much more than the average.

• Total event size depends upon the physics.
  – Accidental hits unconnected with event.
  – Electronic noise.
  – Distribution of event sizes.
    – Some very large events.
Data Acquisition for Big Experiments

- Detecting hardware can be large (many thousands of channels) and physically distributed within the detector. So we need to:
  - Tag where the data came from (and when).
  - Gather all data “fragments” for one Event together.
  - Do it as efficiently as possible (no back-end dead time)

- Experiments typically run for weeks or months. So we need:
  - Stability.
  - Control - to start and stop the whole system.
  - Monitor all experimental conditions under which data was taken.
Anatomy of a DAQ System

- **Readout** – Digitizing detector signals
- **Triggering** – Choosing the data we want to keep
- **Data formatting** - Standardize the data we are saving
- **Event building** - Putting all the event fragments together
- **Event transport** - Make events available to all
- **Event storage** - Save events for later analysis
- **Run Control** - Configure, start and stop experiments
- **Monitoring** - Tell the experimenter what’s going on
Detector Readout Example - a scintillator

- A particle deposits energy in a scintillating material that converts it into light.
- A Photo Multiplier Tube (PMT) converts the light into a pulse of electricity.
- The charge is captured to generate a voltage.
- These pulses are typically fast (~10-20 ns wide)
- An Analog to Digital Converter (ADC) measures the voltage as a digital value.

![Diagram showing the process of scintillation and detector readout]
Picture of test scintillators in Hall-D
Array of scintillators
Sampling vs Integration - ADCs

- A traditional “integrating” ADC can take many microseconds to digitize a pulse. A gate (logic) pulse, generated by trigger electronics marks the region of interest for the signal and enables integration of the charge from the signal pulse.

- This type of ADC generates a single measurement representing the charge sum during the gate.

- During the digitization period after the gate any later pulses from the detector signal are lost (readout dead time).

- A Flash ADC samples continuously at a fixed rate based on an input clock.

- For example, a 250 MHz ADC samples every 4 ns and generates ~5-15 measurements during a typical phototube signal gate.

- These samples describe the pulse shape as well as the total charge.

- There is NO dead time!
Front-End Electronics (Modules / Buses)

- Big Detectors in the experimental halls can have many 1000s of channels. All must be digitized (ADCs TDCs).
  - Pack these circuits onto modules
  - Pack modules into Crates
  - Place crates into Racks
  - Use a standard bus to connect everything to a CPU.

- Common parallel bus standards still found in experimental nuclear physics include:
  - CAMAC (24 bit, 3MB/s)
  - FASTBUS (32 bit, 40MB/s)
  - VME (32/64 bit, 40 MB/s – 320 MB/s)
  - PCI (64 bit, 500 MB/s)

- In the last 15 years or so, parallel buses have been mostly replaced or extended using serialized buses.
  - E.g. USB, PCIe, VME/VXS
  - We will talk more about this later…
We can now buy programmable logic chips that allow us to implement complex algorithms in the firmware on a single chip.

Virtually all the hardware signal processing can be done in these very powerful chips. We just have to get all the relevant detector signals to them.

Another important development in electronics is the **ASIC** or Application Specific Integrated Circuit.

It is similar to an FPGA but is generally designed to a much more specific task. It is being used increasingly as an interface to the analog signals from the detectors.
**JLAB – 64 channel Pipeline TDC**

Hit times are measured relative to the Stop pulse (which can also be a Trigger).
This module is currently used in all 4 Experimental Halls at JLAB. It is a core component to establish both Trigger Processing and Data Readout from the detectors.
A Simple Trigger

• How do we know the detector signal came from an event?
  – Fortunately we have more than one detector.
  – Combine data from different detectors to characterize events.
  – Determine which events are interesting.
• Example - Coincidence trigger – two detectors have data within a time window.

This “trigger” can now be used to form a gate for the ADC signals.

But how to match up the trigger with the ADC signals...
An analog trigger

- It takes some time for the trigger logic to decide if a signal should be digitized.

- The analog signal must be delayed so that the gate and signal arrive at the ADC at the same time. Typical coax cables ~1 ns/ft so you could simply delay the signals using long cables.
  - Matching cable lengths is very important.
  - The ADC cannot process a new signal until it is read or cleared.
  - This limits the accepted trigger rate.
Here’s the long cable in Hall-A.

When you have a 1000 channels and multiple logic stages for the trigger. You need A LOT of cable delay for the signals.

Where do you put it...
Replacing all that cable with digital memory.

In a pipelined system a Flash ADC digitizes at a constant rate and stores the values in a memory. Values are clocked into memory at the same rate as the Flash ADC clock which, in the case of the JLAB FADC, is 250MHz (4 nS).

For example, if the trigger logic takes 200 nS we know that trigger corresponds to measurements 200/4 = 50 clocks down the memory pipeline.

The readout software can Read all the samples or calculate the integrated charge (sum the samples) + save the time of arrival.
Trigger logic evolves

- Triggers used to use a lot of electronics wired together. We can’t do that now:
  - Propagation time through cables limit trigger rates.
  - Modern experiments require very complex trigger decisions.

What if we consider using FPGAs and high speed serial links?

Hall C – Trigger processing - Circa 1994
VXS Platform (VME + Switched Serial)

- JLab standardized on this technology for the 12GeV Upgrade
- Originally used for the L1 trigger data path
- Serial uses Dual Star topology
- Up to 20Gb (4 lanes) from each Payload to the 2 Switch slots (A, B)
- Easy distribution of Trigger or low jitter clock to all modules in the crate.

And there are multiple payload options....

Trigger Processor in a switch slot

FPGAs

JLAB Sub system Processor (SSP) (FPGA board)
Example – The GlueX trigger

• Each ADC sends signals to a crate level trigger processor over VXS serial bus.
  • And stores the signal locally in memory
• Each CRATE Trigger processor sends signal to a global trigger for all crates over fiber optic cable.
• Global trigger tells trigger supervisor (TS) which events are good.
• The TS tells Trigger Interface (TI) board in each crate.
• The TI signals the CPU to read out crate and provides information about which trigger the data belongs to.

Copy data from all ADCs in the crate To the CPU
GlueX trigger, starts at Flash ADC crate

FADC Boards are connected to CPU via the VME bus.

16 channels per board

Intel CPU Read Out Controller (ROC) running Linux

15 boards = 240 channels per crate

Trigger interface

FADC Boards are connected to CPU via the VME bus.

FADC trigger data sent over VXS serial bus

Trigger Processor Board sends local trigger info to the global trigger over fiber

15 boards = 240 channels per crate
All crates connect to the Global Trigger Crate

- Intel CPU controller
- Sub-system processor board (SSP)
- Eight boards with eight connectors each so up to 64 crates.
- Global trigger processor (GTP)
- Outputs to trigger supervisor
JLAB Clock and Trigger Distribution System

**TS** – Trigger Supervisor (VME/VXS)
**SD** – Signal Distribution Board (VXS)
**TD** – Trigger Distribution (VME/VXS)
**TI** – Trigger Interface (comes in several flavors)

The TI board can be used as a TS for small (up to 9) front-ends (ROCs)

For large DAQ systems with many front-ends (ROCs)

The TI board can be used as a TS for small (up to 9) front-ends
Final trigger goes to Trigger Supervisor crate

- Trigger Distribution board (TD).
- Optical trigger link back to crates.
- Intel CPU for control and configuration of the TD and TS.
- VXS serial backplane
- Signal Distribution board (SD)
- Trigger Supervisor (TS)
Then back to the start to trigger VME readout

- The TI board gets from the trigger supervisor:
  - Signal to CPU to read the memory of the ADC boards.
  - TI is read to get data about which events the ADC data belong to.
- The CPU:
  - Copies ADC and TI data into memory via VME backplane,
  - CPU encodes data in EVIO data format.
  - Sends the data over the network to the rest of the data acquisition system.
CODA – CEBAF Online Data Acquisition

• What is CODA (also see coda.jlab.org)
  – Software toolkit for implementing data acquisition systems.
  – Hardware/Electronics
    • Custom boards like Trigger, TDCs and ADCs.
    • Support for commercial hardware.
  – Software includes:
    • Interface with electronics (libraries/drivers).
    • Readout Front End and format data (ROC)
    • Inter-process communication - Control and Data (cMsg)
    • Merge data streams (DC, PEB, SEB)
    • Give users access to data for analysis and monitoring (ET System)
    • Write data to files (EVIO, ER)
    • Manage and control the data acquisition system (AFECS)

• CODA is modular. Build a single crate DAQ or a full Experimental Hall system
CODA – Support/Documentation

Web: https://coda.jlab.org

Fileserver: /u/site/coda
CODA – Version 3

Diagram showing various components and their connections:

- **AFECS** (Agent Framework Experiment Control System)
- **GUI** (Graphical User Interface)
- **COOL Database**
- **Platform**
- **EMU (Event Builder)**
- **EMU (Event Recorder)**
- **ET (Event Transport)**
- **User**
- **File**

**Hardware Components:**
- **VXS**
- **VME**
- **FASTBUS**

**Abbreviations:**
- **ROC** – Readout Controller
- **EMU** – Event Management Unit
- **ET** – Event Transport
- **AFECS** – Agent Framework Experiment Control System
The CODA ROC is a software application that has multiple options for the User to implement custom code or hardware for their specific needs.
CODA3 – The EMU Component

- **EMU – Event Management Unit** – is a JAVA-based general processing application for DAQ. It comes in many flavors:
  - **DC** – Data Concentrator
  - **PEB** – Primary Event Builder
  - **SEB** – Secondary Event Builder
  - **ER** – Event Recorder
  - **FCS** – Farm Control Supervisor
  - **STA** – Stream Aggregator

- Input/Output Connections made via an ET system or by EMUSocket protocol which is part of the CODA cMsg library (it allows for multiple connections or “fat” pipes on high bandwidth networks)
Run Control - AFECS

The “Platform” is a JAVA-based application running multiple “agents” that monitor and control external CODA client components (ROC, PEB, ER…) or internal processes (scripts). Multiple run configurations can also be running simultaneously.

Many “rcgui” processes can communicate with a single Platform that is defined by a COOL Database with a Name = env(EXPID).

External commands can be used in User scripts to communicate directly with the platform.
Event Transport (ET)

- Allocating and freeing buffers is time consuming.
- The ET system gives programs access to data via pre-allocated shared memory buffers.
- The system uses a railroad metaphor. Empty data buffers originate at Grand Central. They are filled by data producers and tagged to describe the content.
- The buffers “move” around a circular track and at each Station the tag is checked to see if the buffer should stop at the station.
- An event monitor could set up station, S2, to take 1% of the events.
- An event filter could set up S3 to take all events. Discarded events are sent back to GC good ones move on.
- An event recorder (S4) takes all events and, after the data is written to a file sends the buffer back to GC.
The program **jcedit** allows the User to graphically create different DAQ configurations, defining the components, data links, data files and other details.

Here is a simple example for one ROC connected to an Event Builder and the Event Builder is then “throwing the data away”
GLUEX configuration:
50 ROCS, 4 Data Concentrators, 1 Secondary EB, 1 Event Recorder.
Event Building challenges

• The GlueX trigger rate is up to 90 kHz and there are 50+ crates.

• The total data rate is around 1.5 GByte/s.
  – 30 MByte/s average per each of the 50 incoming links.
  – 1.5 GByte/s through the EB, ET and ER.
  – Since data is copied several times the data rate inside machine running EB is several times 1.5 GByte/s.

• This is what is commonly called the “bottleneck” effect. Even if the network bandwidth is sufficient. It could be a lot for one machine to handle.

  – Solution: Multi-Stage Parallel Event Building.
If all the Front-End (ROCs) generate too much data in aggregate - make the event building parallel.

Two Stage Event Building
2 Parallel Data Streams
   (each writes half the data)

Front-End ROCs send to
4 Data Concentrators
2 Secondary Event Builders
2 Event Recorders
   - each writing to multiple files

By default data is routed from DCs to SEBs equally with alternating event blocks.

Control Events go to both streams.
User events will only go to the first stream.
EVIO Primitive Data Structures

- EVIO data formats are based on 32 bit words

**Evio Header Formats**

**Bank**:
- 32 bits
- MSB (31) to LSB (0)
- Length (32 bit words, exclusive)
- Tag (16 bits) (2) Type (6) Num (8)

**Segment**:
- Tag (8 bits) (2) Type (6) Length (16)

**Tag Segment**:
- Tag (12 bits) Type (4) Length (16)

**Evio Content Type Codes**

<table>
<thead>
<tr>
<th>Content Type</th>
<th>Primitive Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0</td>
<td>32 bit unknown (not swapped)</td>
</tr>
<tr>
<td>0x1</td>
<td>32 bit unsigned int</td>
</tr>
<tr>
<td>0x2</td>
<td>32 bit float</td>
</tr>
<tr>
<td>0x3</td>
<td>8 bit char* (string)</td>
</tr>
<tr>
<td>0x4</td>
<td>16 bit signed int</td>
</tr>
<tr>
<td>0x5</td>
<td>16 bit unsigned int</td>
</tr>
<tr>
<td>0x6</td>
<td>8 bit signed int</td>
</tr>
<tr>
<td>0x7</td>
<td>8 bit unsigned int</td>
</tr>
<tr>
<td>0x8</td>
<td>64 bit double</td>
</tr>
<tr>
<td>0x9</td>
<td>64 bit signed int</td>
</tr>
<tr>
<td>0xa</td>
<td>64 bit unsigned int</td>
</tr>
<tr>
<td>0xb</td>
<td>32 bit signed int</td>
</tr>
<tr>
<td>0xc</td>
<td>Tag Segment</td>
</tr>
<tr>
<td>0xd</td>
<td>Segment</td>
</tr>
<tr>
<td>0xe</td>
<td>Bank</td>
</tr>
<tr>
<td>0xf</td>
<td>Composite</td>
</tr>
<tr>
<td>0x10</td>
<td>Bank</td>
</tr>
<tr>
<td>0x20</td>
<td>Segment</td>
</tr>
</tbody>
</table>

Number of unused bytes at end of following data if not a multiple of 32 bits. For shorts, it is 0 or 2. For chars (not strings), it is 0, 1, 2, or 3.
The CODA Readout Controller is responsible for collecting raw data from digitizers as well as trigger and timestamp information and wrapping it up for transport.

There is always a Trigger Bank followed by a User definable number of Data Banks.

Note that Banks can contain Banks or Segments which can further encapsulate information for the User.

Lengths in the headers facilitate finding the start of the next bank or segment.

Data Banks contain information from VME modules or digitizing electronics.
The image shows a screenshot of a software interface titled "EVIO File Viewer - jeviodmp". The interface appears to be a file viewer tool with a focus on EVIO files, which are likely used in scientific data analysis. The tool includes a tree structure on the left side, showing the hierarchical organization of the data, and a table on the right side, displaying specific data values.

The tree structure on the left seems to represent a directory or file system, with nodes labeled as "BANK" and "SEGMENT". The table on the right contains columns labeled "structure", "tag", "data type", "number", "length", and "description". The data displayed includes various hexadecimal values, indicating numerical data points.

The software appears to be used for analyzing or visualizing EVIO files, possibly in a scientific or research setting. The interface is designed to provide a detailed view of the data structure and values, allowing for in-depth analysis and manipulation of the file content.

Overall, the image reflects a sophisticated tool for handling complex data files, particularly those related to scientific research, and provides a comprehensive view of the data's organization and content.
Data storage

• 6 GeV experiments ran at tens of MB/s.
• 12 GeV experiments, hundreds of MB/s.
• Generate tens of petabytes per year.
• Tape is cheap but disk is faster.
• We write data to disk then copy from disk to tape later.
  – Tape speed only needs to handle average rate over a 24 hour period.
  – Tape drives and library robots are expensive and fragile. Writing to disk allows data taking to continue if the tape system breaks.
  – We typically have enough disk to hold several days of raw data.
What is next?

• New experiments are being proposed – they do not fit traditional DAQ
  • Some detectors that do not play well together due to timing.
    • Traditional trigger and event builder strategies are not ideal.
  • Detectors with peculiar topologies.
    • Detectors split or segmented in a way that makes forming a trigger hard.
  • High event and/or data rates.
    • Particles from more than one event in a detector at the same time – need to disentangle.

• The data acquisition from these experiments does not fit well with current techniques.

• In the DAQ business we are always looking at ways to take advantage of the trends in electronics and computing.
What would we expect to be happening?

• If we extrapolate current trends:
  – CPUs are becoming more powerful but the performance that matters for online systems is achieved mainly through doing more in parallel rather than improvements in per-core performance.
  – FPGA performance, affordability and usability are still improving.
  – IO and serial network bandwidth still seem to be growing exponentially.

• It is time to revisit the ideas that have dominated nuclear physics detector readout for the past twenty or thirty years.
  • Move things that we moved from hardware to software 25 years ago BACK into hardware (or at least firmware).
  • Move away from parallel busses – serial links are faster and more cost effective.
  • Reevaluate data flow management.
In traditional triggered readout:
- Data is digitized into buffers and a trigger, per event, starts readout.
- Parts of events are transported through the DAQ to an event builder where they are assembled into events.
- At each stage the flow of data is controlled by “back pressure”.
- Data is organized independently by event.

What if we can get rid of the the whole Front-end Trigger Processing path?

In a Streaming readout:
- Data is read continuously from all channels.
- Validation checks at source reject noise and suppress empty channels.
- The data then flows unimpeded in parallel channels to storage or a local compute resource.
- Data flow is controlled at source.
- Data is organized in multiple dimensions by channel and time.
Streaming advantages

• The lack of a trigger means:
  – Potentially interesting physics is not discarded.
  – Run groups of experiments in parallel.
  – The front-end system is simplified.
  – Readout speed is independent of detector response time.
  – Flow control at individual data sources not via backpressure.

• Parallel streams mean:
  – Avoid event building bottlenecks
  – DAQ is robust against minor hardware or firmware glitches.
  – Can use different analysis techniques depending on the detector systems rather than just reconstructing events.

• Requires robust and accurate time stamp generation and distribution.
  – Is still a simpler task than an online trigger.
  – Already have a clock distribution with the trigger anyway
In practical terms, what does it look like?

- A timing system synchronizes streams at source.
- Front-end outputs data in streaming format on a net.
- DAQ consists of tiers processing separated by tiers of temporary storage.

Software processing ultimately reduces the final data storage requirements:
- A “high Level” or Software “trigger”
Streaming data can be thought of as Triggered mode where the trigger is a fixed pulser and you keep all the data for a single channel generated from the last pulse.

A 250 MHz FADC generates a 12 bit sample every 4ns. That's 3 Gb/s for one channel. A 16 channel module is 48 Gb/s. That is over twice the available VXS bandwidth. But we don’t need ALL the data.

Within the FPGA we keep only the data around a Region of Interest (ROI) from each channel, along with a fine time stamp in each time slice window (65µs).

Depending on hit rates and available bandwidth, we can keep the individual samples or just compute a sum.

Note: The JLAB FADC can simultaneously operate in “triggered” mode with an 8µs pipeline and 2µs window.
FADCs - Triggered vs Streaming

**Triggered Mode**
- ROI
- External Trigger (timestamp)
- PL (0-8µs)
- PTW (0-2µs)

**Data we get on a trigger:**
- FADC waveform values for the ROI
- Threshold Sample # (hit time)
- Trigger absolute time stamp

**Streaming Mode**
- Time Slice (Frame # & timestamp)
- ROI
- TH

**Data we get for a Frame:**
- Pedestal subtracted sums over an ROI for every hit over threshold
- Threshold sample # fine time stamp for each hit
- Frame # and absolute time stamp for the frame

**ROC Data Format**
- Trigger Bank
- Tag
- DT
- M
- Trigger # timestamp

**Stream Info Bank**
- Tag
- DT
- SS
- Frame # timestamp
Two ROCs are defined for the DAQ System:

**VME_ROC**
- Runs on the CPU
- Reads out Triggered data

**VTP_ROC**
- Configures and manages the stream data

**EMU***
- Event Builder
- File
- CPU

**FADCs**
- 10Gbps

**VXS Crate**
- phototube
- scintillator
- e-

**DISC**
- Trigger

**PC/Server**
# Beam Tests

**Calorimeter spectra – Streaming Data**  
(Electron beam centered on the central crystal)

<table>
<thead>
<tr>
<th>ERSAP</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Graphs" /></td>
<td><img src="image2.png" alt="Graphs" /></td>
</tr>
</tbody>
</table>

**Time Slice Frame containing a Trigger**

ERSAP: channel vs hitTime (on indra-s1)

- Can look for “triggers” in software later

![Graph](image3.png)
Beam Tests cont...

- Rates were relatively low for these tests
- The electron beam varied in current depending on if they were filling the PETRA synchrotron.

- Note the respective data rates on Run Control.
  - Streaming frame and data rates are relatively constant (15.2kHz, ~1.3MB/s)
  - Triggered rates rise and fall as the electron beam comes and goes (reading out wave forms even in triggered mode generates a lot of data)
VME Readout is a bottleneck for Streaming

The JLab VTP board is able to read 250 MHz FADC via the VXS serial backplane and stream the data out over the front panel transceivers.

a) Read a crate in this mode.
   - Implement Zero suppression logic
   - Flow control
   - What comes out on the front panel fiber?
   - How to interface with DAQ?

b) The VXS interface allows read out for many detectors in streaming mode!

One Question. Can we repackage to remove need for expensive VXS crates etc. for some or all the detectors?
   - Cheaper testing systems for university groups
Recent Front-End Electronics developed by the FEDAQ group have been implemented in several detectors (CLAS12 RICH and GlueX DIRC).

Currently the detector data is sent to another FPGA board in a VXS crate, but in practice the optical link off the FEE FPGA can be streamed directly to an ethernet port on a PC.

This can facilitate development by collaborators without big system overhead, but it is also a peek into the future streaming model for Big experiments.
**JLAB VXS Architecture**

- **Raw Stream data from modules**
  - FADC (16 Channels)
  - Up to four 10Gb Ethernet links

- **Aggregate Stream Blocks (from payload modules)**
  - Up to 16 Modules (VXS 10-20 Gb/s)
  - FPGA

- **Aggregate Streams (including other crates)**
  - CODA_SRO Software Aggregation Interface to Back-End

**JLAB (Crateless) Architecture**

- **Up to 48 links**
- **1Gb Ethernet**
- **CODA_SRO Software Aggregation Interface to Back-End**

- **Custom serial fiber link**
- **Up to four 10Gb Ethernet links**
- **Up to four 10Gb Ethernet links**
Summary

- Data acquisition is constantly challenging.
  - Technology changes all the time.
  - Physicists think up experiments with tougher requirements.
  - The boundary between hardware and software is fluid and depends on what is available when a system is implemented.
  - There is always some R&D happening for custom hardware and to discover new algorithms, techniques.
  - To be honest, we do it because it’s fun, and we get to play with all of the cool toys (and design some of our own)
  - Thank you for your attention