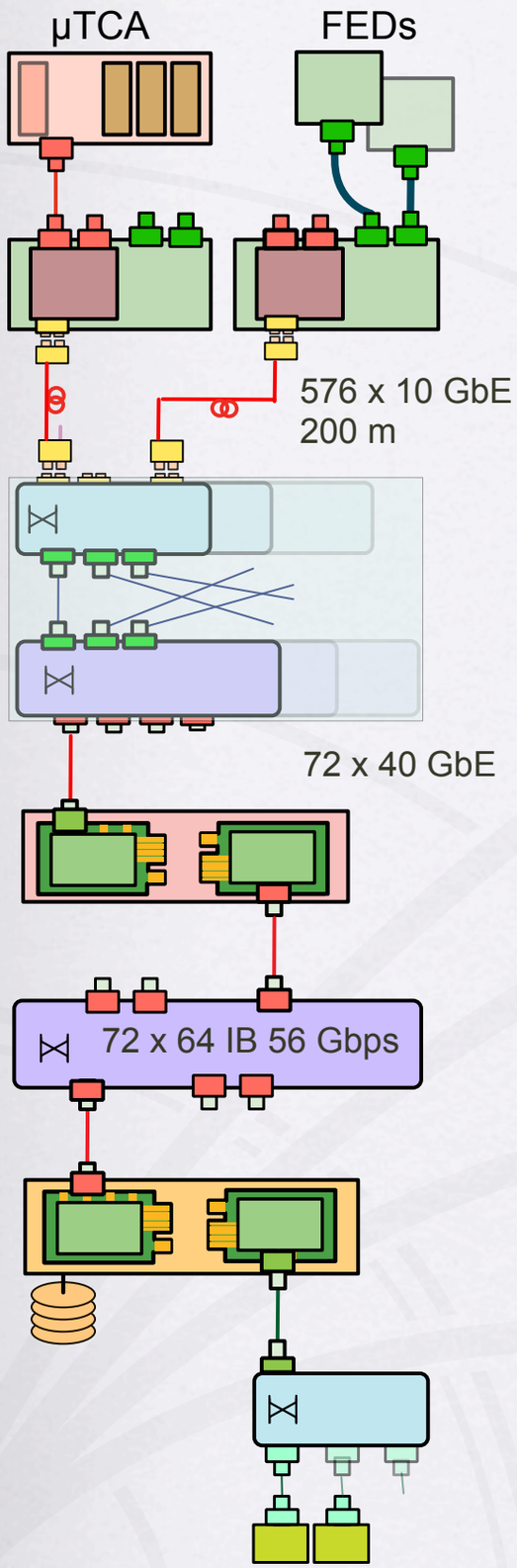
Verification Steps for the CMS Event-Builder Software

Remigius K Mommsen
Fermilab

CMS Event Builder



Detector front-end (custom electronics)

- ~700 front-end drivers (FEDs) with ~2kB/fragment at 100 kHz

Front-End Readout Optical Link (FEROL)

- Optical 10 GbE TCP/IP

Data Concentrator switches

- Data to Surface
- Aggregate into 40 GbE links

72 Readout Units (RUs)

- Combine FEROL fragments into super-fragment

Event Builder switch

- Infiniband FDR 56 Gbps CLOS network

62 Builder Units (BUs)

- Event building
- Temporary recording to RAM disk

Filter Units (FUs) (~16k cores in ~900 boxes)

- Run HLT selection using files from RAM disk
- Select O(1%) of the events for permanent storage

Event-Builder Software

Event builder is part of the CMS Online Software suite (c.f. talk from Luciano)

- C++ software compiled with gcc 4.4.7 (no C++11)
- ~15k lines of code (excluding the framework)
- Controlled by CMS run-control via SOAP messages

Consists of 3 applications derived from same templated base class

- 1 event manager (EVM)
  - Orchestrates the event building
  - Receives the trigger information
- 71 readout-units (RUs)
- 62 builder-units (BUs)

O(40) threads per application for different tasks and to parallelize tasks

- All threads are pinned to a given CPU core

Optimized for I/O performance

- Threads & memory located close to Ethernet NICs
- Data transfers over Infiniband uses RDMA
- Code for checking event integrity is CPU limited

Versatile Code

Event-builder (EvB) not only used in production

- fedKit stand-alone application for lab-bench use
  - Same code with different XML configuration
  - Controlled by user-friendly python script
- miniDAQ system to readout parts of subsystems
  - Small scale version of production system
  - Used for testing, debugging or calibrating subsystems
  - 6 independent setups separated from central DAQ
- Local DAQ systems maintained by sub-system groups

Provide data integrity checking and error reporting

- Extensive checking of the data
- Ability to dump events to disk at various stages
- Detailed error reports

Testing Procedure

Testing of new a new version goes over multiple steps

- Unit tests
- Stand-alone test cases
- System integration test bed (daq2val)
- Production system

Unit Tests

Standalone C++ applications, i.e. not using any testing f/w

Mostly used for testing algorithms

- Is the correct value returned especially for edge cases?

Test critical & well isolated parts

- Run many times to catch memory corruption or data races

Pros:

- Easy to debug and profile

Cons:

- Works only for isolated part of the code
- Virtually impossible to test interplay of applications

Stand-alone Test Cases

A small setup on a single machine

- 1 EVM & 0-2 RUs & 1-4 BUs
- Dummy data is generated inside the applications or with a separate application emulating the front end

Based on python scripts (~1000 lines of code)

- Generation of XML configurations to setup the test case
- Start and stop the XDAQ applications
- Emulate a simplified run control environment
- Drive the system through different scenarios
- Check states and parameters of applications

~50 test cases implemented (~2000 lines of code)

- Behavior using different settings
- Emulate failures and edge cases
- Running all tests takes ~30 minutes

Example of a Test Case

Check that EVM goes into SyncLoss state if data is skipped

- Define a configuration with 1 EVM and 1 RU, each with 4 dummy FEROLs as input, and one BU
- Run the test by starting the system, skip an event, and check that application states are okay and the event was dumped to a file

```
def fillConfiguration(self,symbolMap):
    evm = RU(symbolMap,[
        ('inputSource','string','Socket')
       ])
    for id in range(0,4):
        self._config.add( FEROL(symbolMap,evm,id) )

    ru = RU(symbolMap,[
        ('inputSource','string','Socket')
       ])
    for id in range(4,8):
        self._config.add( FEROL(symbolMap,ru,id) )

    self._config.add( evm )
    self._config.add( ru )

    self._config.add( BU(symbolMap,[
        ('dropEventData','boolean','true'),
        ('lumiSectionTimeout','unsignedInt','0')
       ]) )
```

```
def runTest(self):
    self.configureEvB()
    self.enableEvB(runNumber=1)
    self.checkEVM(8192)
    self.checkRU(8192)
    self.checkBU(16384)

    print("Skipping an event on FED 2")
    self.setAppParam('skipNbEvents','unsignedInt','1','FEROL',2)
    time.sleep(5)
    self.checkAppState("SyncLoss","EVM")
    self.checkAppState("Enabled","RU")
    self.checkAppState("Enabled","BU")
    self.checkAppParam('eventRate','unsignedInt',0,operator.eq,"EVM")
    dumps = self.getFiles("dump_run000001_event[0-9]+_fed0002.txt$")
    if len(dumps) != 1:
        raise ValueException("Expected one FED dump file, but found: "+str(dumps))
    self.haltEvB()
```

Running the Test Cases

2 modes of running the tests

- Test case can be run individually in an interactive mode
  - Mostly useful for debugging
- All test cases are run automatically and logged
  - Done after any significant change to the code

Tests are independent of the XDAQ build system

```
2x1_logNormal                : 09:52:29 09:52:39  Passed
2x1_mismatch                 : 09:52:39 09:53:30  Passed
2x1_mismatch_ptfrl           : 09:53:30 09:54:20  Passed
2x1_multiFEDs                : 09:54:20 09:54:30  Passed
2x1_preallocate              : 09:54:30 09:54:41  Passed
2x1_rateLimit                : 09:54:41 09:55:11  Passed
2x1_singleRequest            : 09:55:11 09:55:21  Passed
2x1_smallBlockSize           : 09:55:21 09:55:32  Passed
2x1_write                    : 09:55:32 09:56:33  Passed
2x2_cloud                    : 09:56:33 09:56:57  Passed
2x2_failBU                   : 09:56:57 09:58:07  FAILED  ValueError: EVM claims 332198 events were built, while BU count gives 332218 events
2x2_lsLatency                : 09:58:07 09:58:27  Passed
2x2_quarantined              : 09:58:27 09:58:41  Passed
2x2_stale                    : 09:58:41 09:58:55  Passed
3x1                          : 09:58:55 09:59:11  Passed
```

Pros & Cons of Test Cases

Pros:

- Very flexible to test the code under various situations
- Writing a new test case before the code asserts its indented behavior
- Assures that changes do not break other parts
- Some test cases reproduce an error seen in production

Cons:

- Cannot test interfaces with the outside world
  - E.g. run control or monitoring
- No tests of performance
  - Tests can be run on multiple machines to measure performance
  - Not used so far
- Reliably reproducing race conditions virtually impossible

System Integration Test Bed

Small scale version (~5%) of the production system

- Uses the same hardware versions
- Has all XDAQ services
- Uses the full run control and configuration framework
- Detached from production system

Pros:

- Allows to test the interaction with the other components
- Can be used to assess code performance
  - Measure the overall performance
  - Inspect running code with perf

Cons:

- Too small to see any scaling issues
- Very limited ability to test error scenarios
- No automatic testing

Production System

The ultimate testing environment

Pros:

- Full scale tests of scaling and performance
- Real detector data spans all cases of failures

Cons:

- Limited availability for tests
- Failures at this stage quickly translate into lost luminosity

Summary

The event-builder s/w is a critical part of the CMS DAQ system

- Any failure translates into lost luminosity
- Interacts with many other components
- Tool to commission and debug front-end readout (h/w & f/w)

Testing is done on several scales

- Unit tests to test cases up to full scale tests with the production system
- Possible improvements for the future
  - Automatic testing in daq2val test-bed with failure scenarios
  - Automated and regular performance measurements
  - Use a testing framework for some or all steps?