FTK system issues
part 2

Alberto Annovi

Why SVT succeeded

Pulsar has similar concepts

- Performance:
  - Parallel/pipelined architecture
  - Custom VLSI pattern recognition
  - Linear track fit in fast FPGAs now with DSP blocks
  - FTK adds: variable resolution, 2 steps, pattern from constants
- Reliability:
  - Easy to sink/source test data (many boards can self-test)
  - Modular design; universal, well-tested data link & fan-in/out
  - Extensive on-crate monitoring during beam running
  - Detailed CAD simulation before prototyping
    - See poster by Mircea Bogdan
- Flexibility:
  - System can operate with some (or all) inputs disabled
  - Building-block design: can add/replace processing steps
  - Modern FPGAs permit unforeseen algorithm changes
- Key: design system for easy testing/commissioning

Original talk: http://www.pg.infn.it/beauty2005/talks/beauty05-m.dellorso.pdf

Modularity …

- SVT had a single connector (except fibers from silicon)
- We already have Slinks from ROD and to ROS
- Slink provides:
  - LINK_UP handshake,
  - error control
  - possibility to exchange control words in both directions:
    - FREEZE on ERROR
    - FREEZE on LVL1 tag
    - Future possibilities

Can we use slinks every where?

- Special cases (what is planned?):
  - DF → DF inter board/crate communication
  - DF → AUX many data links one return channel
    - Two destination AUX: use one or two return links?
  - AUX → final card (exceeding 40MHz word rate)
- Need one extenstion (?):
  - multiple data stream with one return stream
- Advantages:
  - Write and debug one firmware for all boards
  - Have common sw routines for initialization, monitoring, …

S-Link extension for FREEZE

- Slink HW specs
  - http://hsi.web.cern.ch/hsi/s-link/devices/hola/hw_spec.html
- Will need to involve Slink experts
- Forward channel easy:
  - protocol can send control words between events
  - Can we send control words in the middle of one event? Is this already a feature?
- Return channel:
  - Can encode information in 4 return lines
  - Used for FTK_IM → dual HOLA (XOFF enable/disable signal)
  - Usable for also freeze but clumsy
- Forward channel already mixes data and commands
- Can we make the SLink symmetric?
  - It would allow to send commands through the return channel
  - NOT an easy extension!
  - Would also make the Slink full-duplex when needed

Slink forward channel

n# IDLEs	NCW	Control Word	3 IDLEs	< 4KB data	CRCC	CRC Checksum	3 IDLEs	NCW	Control Word	n# IDLEs

Table 3. Transmission example

Now used for “begin of fragment”, “end of fragment” control words.
Interpreted by user logic.

Slink return channel

Transmit one of two possible words (16bits)

Bits	Description
[15..0]	IDLE (<K28.5,D5.6> or <D28.5,D16.2>)

Table 4. IDLE frame in return channel

Bits	Description
[1..0]	RL[0]
[3..2]	RL[1]
[5..4]	RL[2]
[7..6]	RL[3]
[9..8]	XOFF# - Flow Control
[11..10]	LDC down
[13..12]	Remote reset
[15..14]	Reserved - Set as 0, ignore on reception

Table 5. Commands in return channel

4 bits out of Slink FW module

This is what we have within Slink protocol

Timing

- Timing information will be important
- Minimum information
  - Arrival time before FIFO of first-last event word
  - FIFO extraction time of first-last event word
  - time when first-last word is sent out of board
  - 4 number / input stream, 2 numbers / output stream
  - Can use dedicated FW registers, probably a RAM to accumulate recent events (next slide)
- Additional (optional) information
  - Write timing (clock counter) in the spybuffers for each word
  - Usually not needed: we could enable it when needed via FW register
- Timing reference:
  - Either use LVL1 accept signal distributed to all boards
  - Or reconstruct delay between boards
  - In any case we need at least one LVL1 accept input for debugging

Timing spy buffer

- Input data arrival time before FIFO

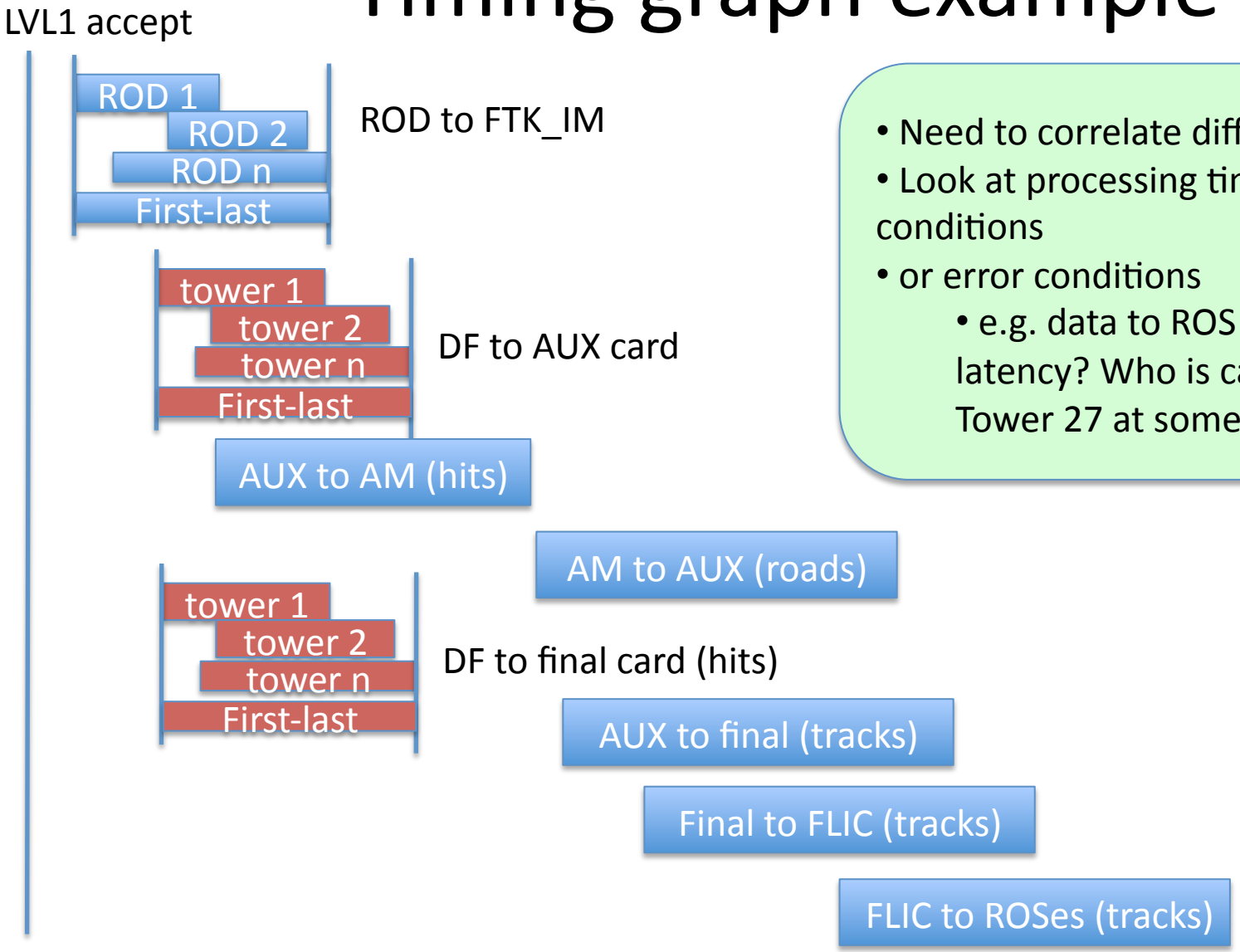
RAM address	LVL1 ID	counter	Last bit
0	0x1234567	5000	0 (first word)
1	0x1234567	5970	1 (last word)
2	0x1234568	6200	0
3	0x1234568	7100	1
….	….	….	…

First event

Second event

- Two spies per input channel (before/after FIFO)
- One spy per output channel
- Typical size 16 events → 32 words
  - Each word 32 LVL1+32 counter+1 bit (or 64 bits with 31 bit counter)
- Need to monitor backpressure
  - Ideas?

Timing graph example



- Need to correlate different streams
- Look at processing time in normal conditions
- or error conditions
  - e.g. data to ROS with 2ms latency? Who is causing this? Tower 27 at some point …

Synchronism

- Where streams merge (DF, AUX, final card, FLIC) we can find data out-of sync. What to do?
- Define valid LVL1 ID (reference stream or majority logic, or simply next ID)
  - "Valid LVL1 ID" is the ID of the event being processed by the board
  - Should raise an error in case "valid LVL1 ID" is ambiguous
    - E.g. two streams report 0x01000100 and two streams report 0x01000101
    - Possible fix use the earliest one as "valid LVL1 ID"
- One stream has LVL1 ID in the future
  - Ignore it (assume empty events) until it aligns
  - Dangerous in case of bit errors..
- One stream has LVL1 ID in the past
  - Discard data until valid LVL1 ID is found
  - Ignore stream in the mean while?
  - Or wait for it until a time out?
- Self recovery options or trigger stopless removal/recovery
- I suggest this logic should be a common VHDL code

Firmware repository

- All firmware (code & bitstreams) should be stored in a CERN SVN repository
  - Also during development: it helps
  - I create this SVN repository: svn+ssh://ftkfw@svn.cern.ch/reps/atlasftkfw
  - Ask me for permission
  - Do you have a preference for a different repository?
  - Let's pick one and move everything there
- SVN repository structure
  - atlasftkfw/BOARD_NAME/FPGA_NAME
- It is not trivial to "version control" a Xilinx/Altera project
- This is what I did for the FTK_IM fpga:
  1. Make an empty SVN directory
  2. Create the project inside it (I tried with Xilinx)
  3. Start adding to SVN all *vhd files
  4. Add to SVN the main project file, and "needed files" (which ones?)
  5. Check out the project in a different dir and try to compile
  6. Remove from SVN temporary files
  7. Add to SVN files missing and needed for compilation
  8. Go back to 5 and iterate. It should converge in a few iterations

board/firmware documentation

- Minimum documentation for each board is
  - Specification of all interface signal/buses/connectors
  - Specification of all externally controllable registers/ memories (e.g. VME address space)
  - Short description of functionality
  - Do we need more information
- I propose to store a latex and pdf file into each SVN board directory
  - Atlasftkfw/BOARD_NAME/doc
- I would keep FPGA fw documentation optional, but code readable and commented

Version control

- Each board should have these registers:
  - **FW Version register**: major 31:16 & minor 15:0
  - Major version will change when software changes as well
  - Minor version change are internal changes
  - **Board serial number** (we will need this)
    - Suggestion: use a small flash to write & store it during board test
    - Also write it on the PCB to be read without power
  - **PCB version register**: to keep track of prototypes
    - Ideally this number is hardcoded to one FPGA (e.g. use 3 pins connected to VCC and GND in the PCB)
- Ideally all FPGAs (or at least those with VME/ATCA access)
  - Should have a **FW version register** + **FW date register**

Common firmware

- Should make a FTK library of common firmware elements
  - SPY buffers
  - Synchronization logic
  - What else?

Online software

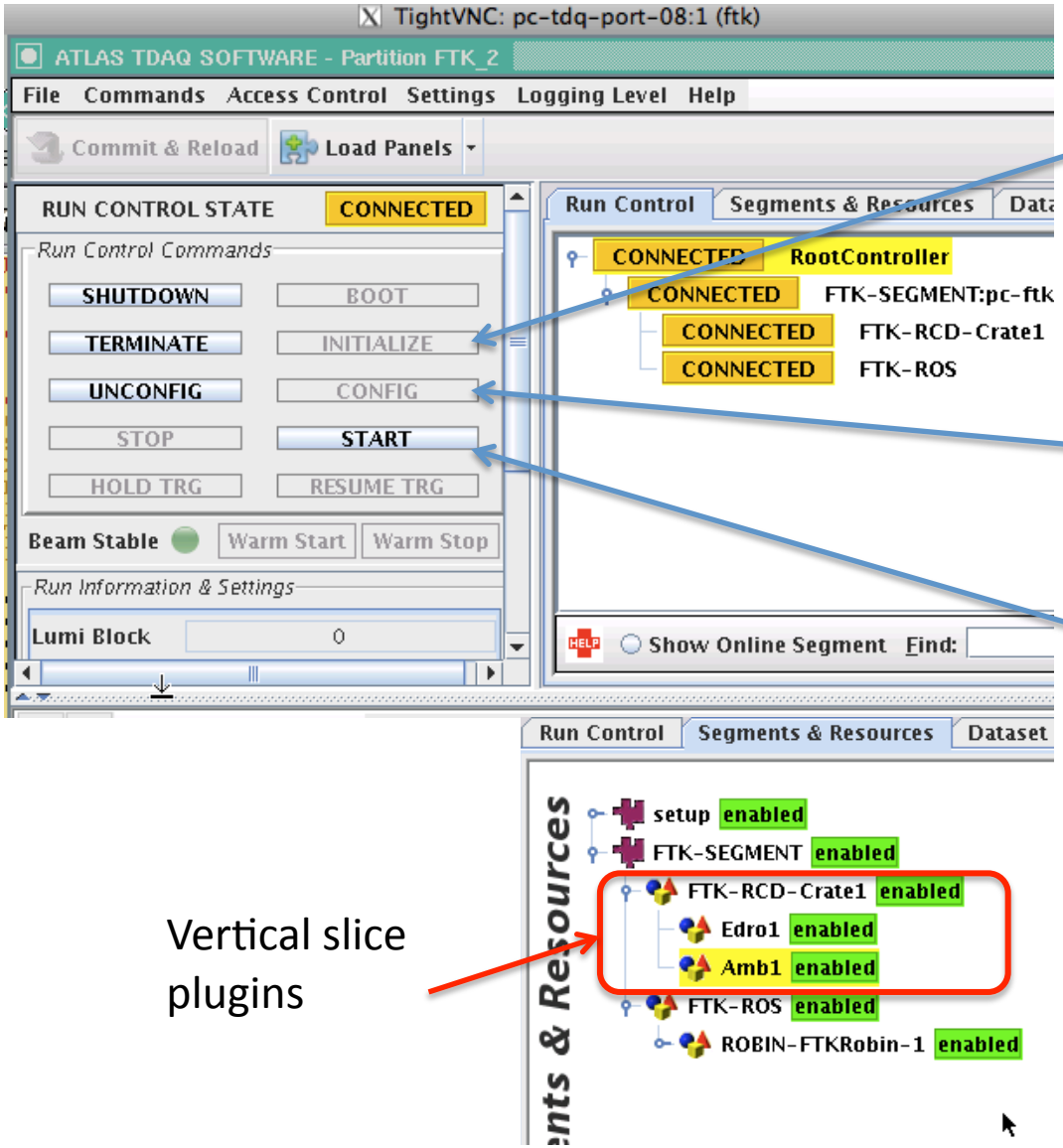
Three goals:

1. HW diagnostic (see Paola's talk last week)
   - We need board level simulation
   - FTKSim should be organized similarly to the HW
     - a set of board-simulation functions called in sequence
     - DF simulation in the wrapper is going in this direction
2. Configuration for data taking
3. Monitoring

Configuration for run

- Run control function needed to configure FTK boards
- Configuration data to be loaded inside FTK
  - Online software will include FTKSim
    - A frozen copy of FTKSim will be imported in the online area
  - Will need to define a function in FTKSim for each memory to be configured providing the data
- For the VS, but likely also for the final FTK:
  - each crate is a rcd application running in the CPU
  - each board is a rcd plugin (RCD = Rod Crate Daq)
  - RCD manual: https://edms.cern.ch/document/577958/1
    - Errata: StartTrigger/StopTrigger methods only for master trigger
- Online code:
  - Now here: svn.cern.ch/reps/vipix/ftk/trunk/ftk_v2
  - Will move to TDAQ repository soon

Run control functions



Vertical slice plugins

- setup()
  - Gets info from OKS
  - Called also at "commit&reload"
- configure()
  - Loads RAMs/registers
- prepareForRun()
  - Currently sends init
- probe()
  - Called periodically
  - Fills IS information

Online monitoring goals

1. Verify data quality
   - Efficiency & noise rejection
   - Use high-level observables: daily Z->ll peak?
2. Verify hardware integrity
   - Compare bit-by-bit with simulation
3. Fast verification
   - Histograms with relevant quantities: # hits, roads, fits, tracks
   - Tower occupancy/efficiency …
4. Timing
   - it is as important as data quality
   - Monitor LVL1accept to end-of-processing
   - Monitor LVL1accpet to start/end-of-processing at each board

This is an important activity that deserves a responsible institution as much as building a board.

Stopless removal/recovery

- Identified trigger conditions for VS
- when FTK output busy (detected in FLIC/ EDRO)
- when no data to ROS (detected by ROS)
- wrong data format to ROS (detected by ROS)
- additional FTK internal monitoring
  - detect problems before they reach ROS
  - React faster

Stopless removal/recovery

- My understanding of it
- Online applications probe the status of the FTK boards and ROS at O(1s) intervals
- If a problem is detected a message is sent to runControl shifter
- runControl shifter activates the stopless removal/ recovery
  - automatic or requires manual acknowledge?
  - Subroutines are called
    - Put the boards in offline mode
      - no data to ROS
      - disable backpressure to RODs
    - Restore working configuration

Offline test stand

- In early SVT days …
- There was a vertical slice (half a crate) always processing data being compared with simulation (low rate)
- provides hot spare always tested and working ready for swap
- The offline test stand will be need
  - Keep it in mind while developing test firmware/ software
  - Again hw/sw modularity will help