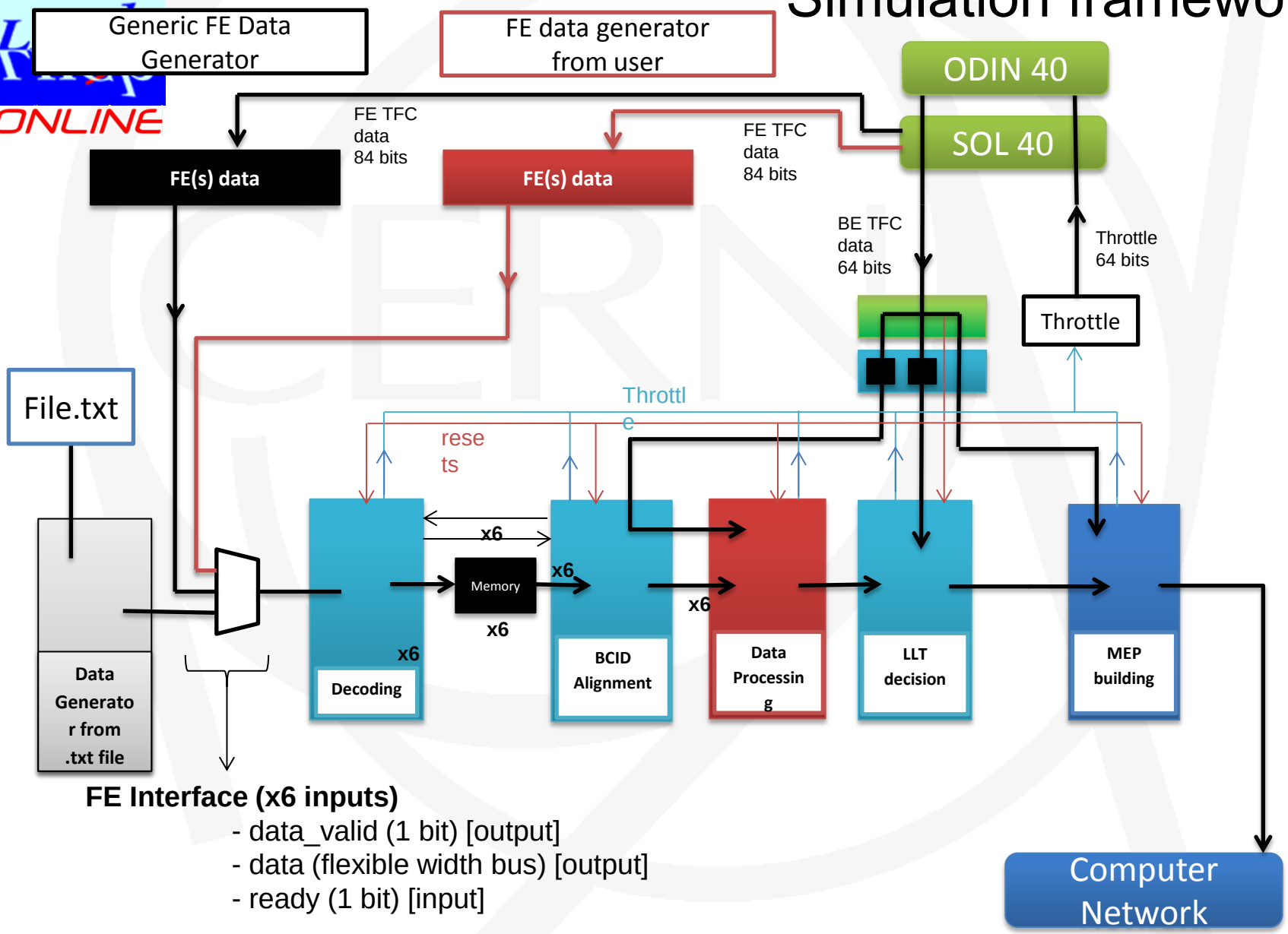
Running simulation for the Mini-DAQ: TFC and FE features

LHCb Electronics Upgrade Meeting

12 December 2013

Federico Alessio

Simulation framework



FE Interface (x6 inputs)

- data_valid (1 bit) [output]
- data (flexible width bus) [output]
- ready (1 bit) [input]

Simulation framework

Philosophy maintaned:

→ flexible, configurable, easy-to-use, collaborative …

Realistic and synthesizable code for TFC + TELL40 + MEP

→ realistic environment

→ follow specs to the very last detail

→ expertise available for it

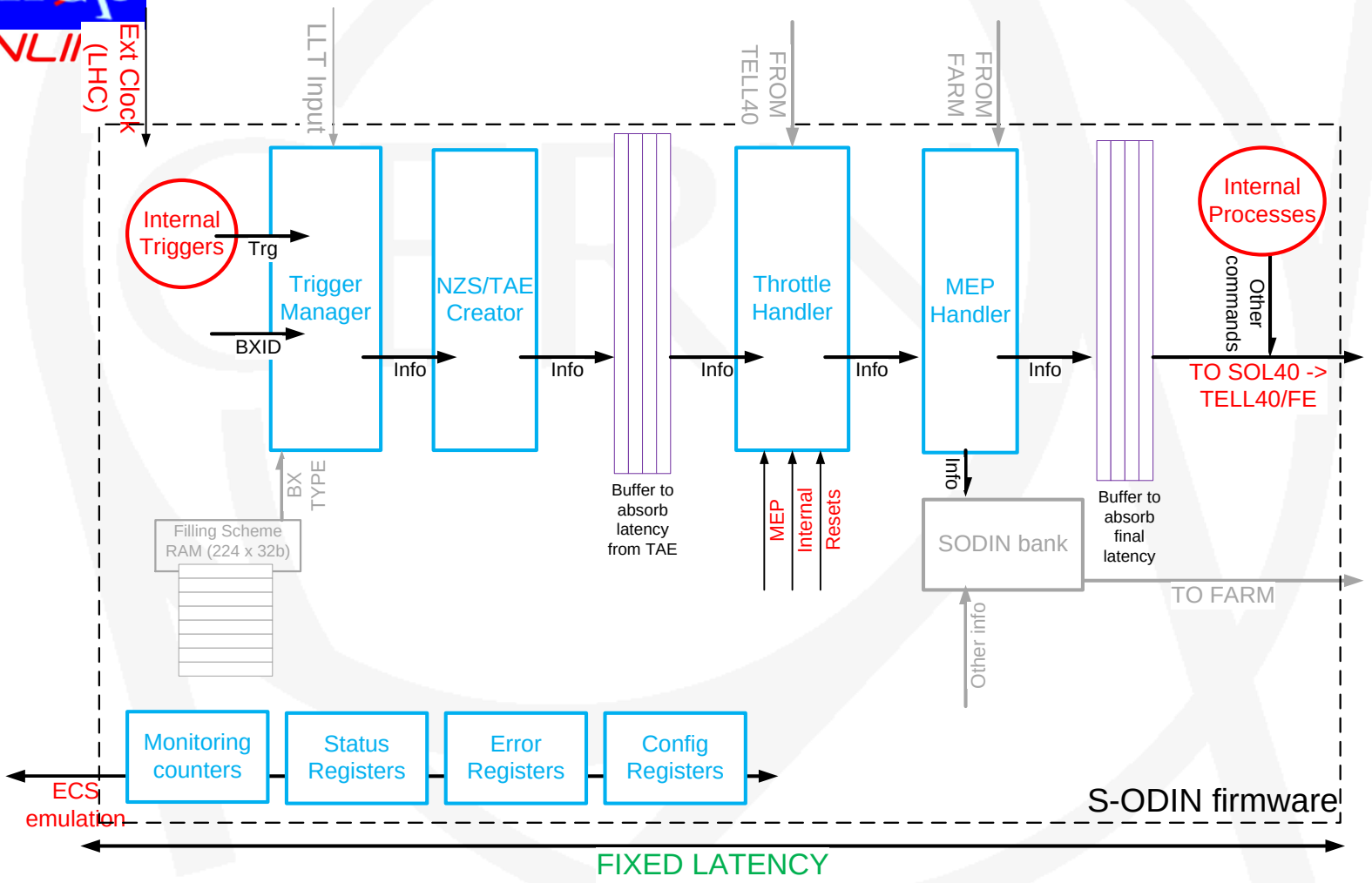
Emulation of different allowed FE encodings

→ generic one

→ from a .txt file (raw data)

→ from you…

S-ODIN HDL code



For details on S-ODIN, see [LHCb-PUB-2012-001]

TFC (fast commands) available

to TELL40

63 .. 52	51	50	49 .. 18	17 .. 14	13 .. 10
BXID(11..0)	Reserve	MEP Accept	MEP Dest(31..0)	Trigger Type(3..0)	Calibration Type(3..0)

9	8	7	6	5	4	3	2	1	0
Synch	Snapshot	Trigger	BX Veto	NZS Mode	Header Only	BE Reset	FE Reset	EID Reset	BXID Reset

to FE

23 .. 12	11	10	9	8 .. 5	4	3	2	1	0
BXID(11..0)	Reserve	Synch	Snapshot	Calibration Type(3..0)	BX Veto	NZS Mode	Header Only	FE Reset	BXID Reset

Periodicity, rates, delays, codes are all configurable
via a simple configuration package

For details on the commands and their usage, see LHCb-PUB-2012-017

Configuration package features I

Everything is explained in the Mini-DAQ handbook document!

```
-- TFC to FE specific configurations parameters
--
constant FE_reset_wait                : std_logic_vector(15 downto 0) := X"00FA"; -- 250 clock cycles
-- number of clock cycles to wait after one FE RESET command

constant NZS_enb              : std_logic                     := '1';
-- enable/disable NZS trigger
constant NZS_consecutive_enb  : std_logic                     := '0';
-- number of consecutive NZS triggers
constant NZS_TAE_wait_clkcycle: std_logic_vector(11 downto 0) := X"005";
-- number of clock cycles to wait after one or more consecutive NZS triggers

-- CALIBRATION TRIGGERS
--
constant CALIBTRG_A_period    : std_logic_vector(15 downto 0) := X"0001";
-- periodicity as a number of orbits (1 = every orbit etc.)
constant CALIBTRG_A_bxid      : std_logic_vector(15 downto 0) := X"0C0F";
-- BXID on which calibration trigger will trigger
constant CALIBTRG_A_enb       : std_logic                     := '1';
-- enable/disable calibration trigger

constant CALIBTRG_B_period    : std_logic_vector(15 downto 0) := X"0001";
constant CALIBTRG_B_bxid      : std_logic_vector(15 downto 0) := X"04AF";
constant CALIBTRG_B_enb       : std_logic                     := '0';
constant CALIBTRG_C_period    : std_logic_vector(15 downto 0) := X"0001";
constant CALIBTRG_C_bxid      : std_logic_vector(15 downto 0) := X"09DF";
constant CALIBTRG_C_enb       : std_logic                     := '0';
constant CALIBTRG_D_period    : std_logic_vector(15 downto 0) := X"0001";
constant CALIBTRG_D_bxid      : std_logic_vector(15 downto 0) := X"020F";
constant CALIBTRG_D_enb       : std_logic                     := '0';
-- same as Calib A
```

Enables NZS triggers and Calibration types

Configuration package features II

```
-- SPECIAL ENABLES
constant SNAPSHOT_enb          : std_logic                      := '1';
-- enable SNAPSHOT command
constant SNAPSHOT_interval     : std_logic_vector(15 downto 0) := X"37B0";
-- number of clock cycles between two SNAPSHOT commands
constant SYNCH_enb             : std_logic                      := '1';
-- enable SYNCH command
constant SYNCH_length          : std_logic_vector(15 downto 0) := X"000A";
-- number of consecutive SYNCH triggers (in clock cycles)
constant SYNCH_wait            : std_logic_vector(15 downto 0) := X"0002";
-- number of consecutive clock cycles to wait after one or more SYNCH commands
constant BX_VETO_enb           : std_logic                      := '1';
-- enable BX_VETO command (ignore if not used in FE)
constant HEADER_ONLY_enb       : std_logic                      := '1';
-- enable HEADER_ONLY command (ignore if not used in FE)
```

Various enables/parameters to emulate TFC commands to FE

Front-End HDL code

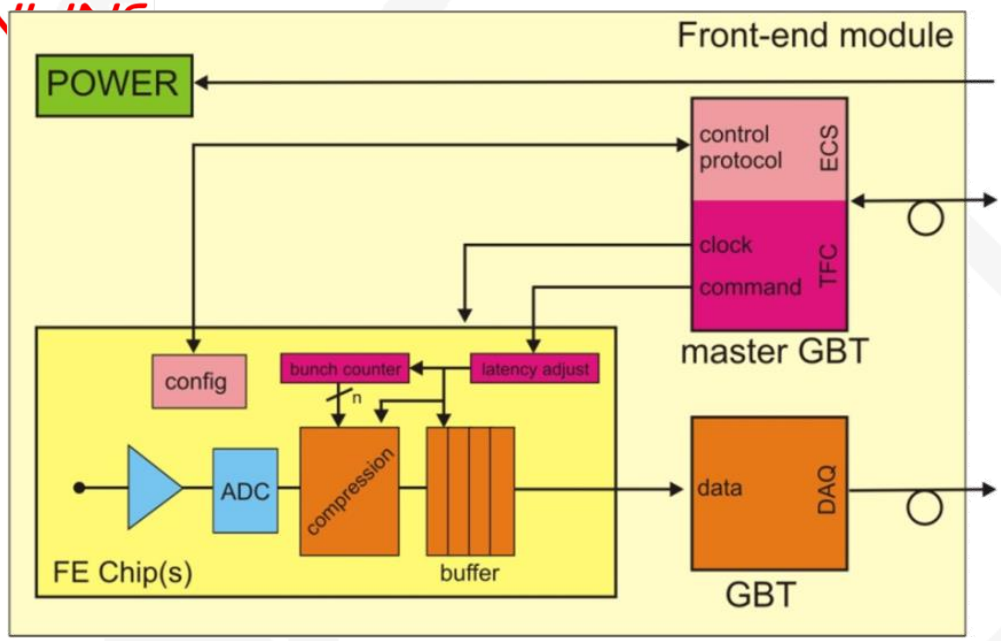
Implemented three generic different types of algorithms to emulate FE data encoding:

- ✓ Variable frame length packing with Variable size header (called VV)
- ✓ Variable frame length packing with Fixed size header (called FV)
- ✓ Fixed frame length packing with Fixed size header (called FF)

NB: this was needed to develop the TELL40 code and study each decoding scenario

For more details, see LHCb-INT-2013-015

Reminder: your (generic) FE



NO TRIGGER to FE!
→ Only commands, clock and slow control

For details, see
LHCb-INT-2011-011

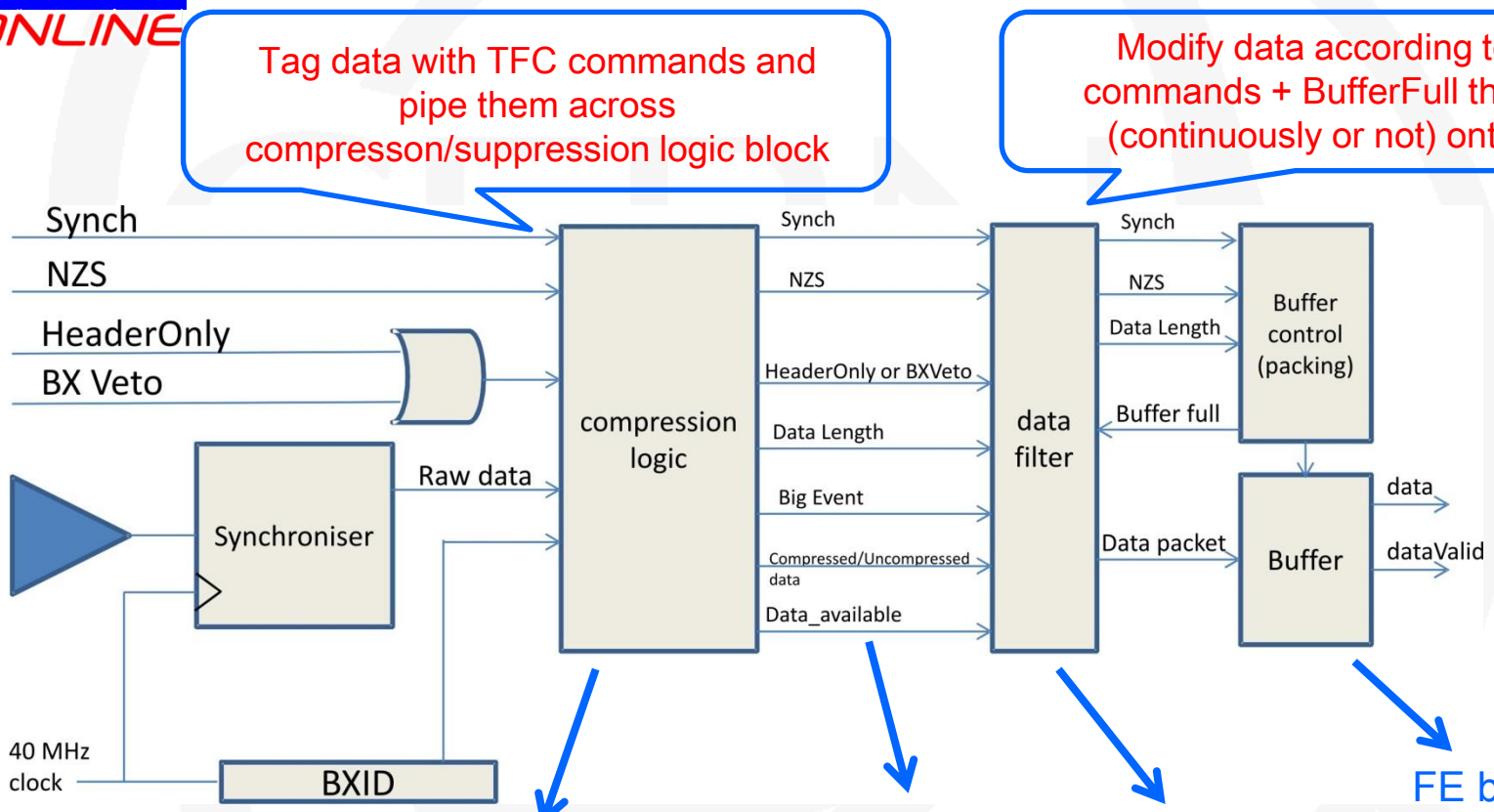
Compress (zero-suppress) data already at the FE

- reduce # of links
- data driven readout (asynchronous) + variable latencies!

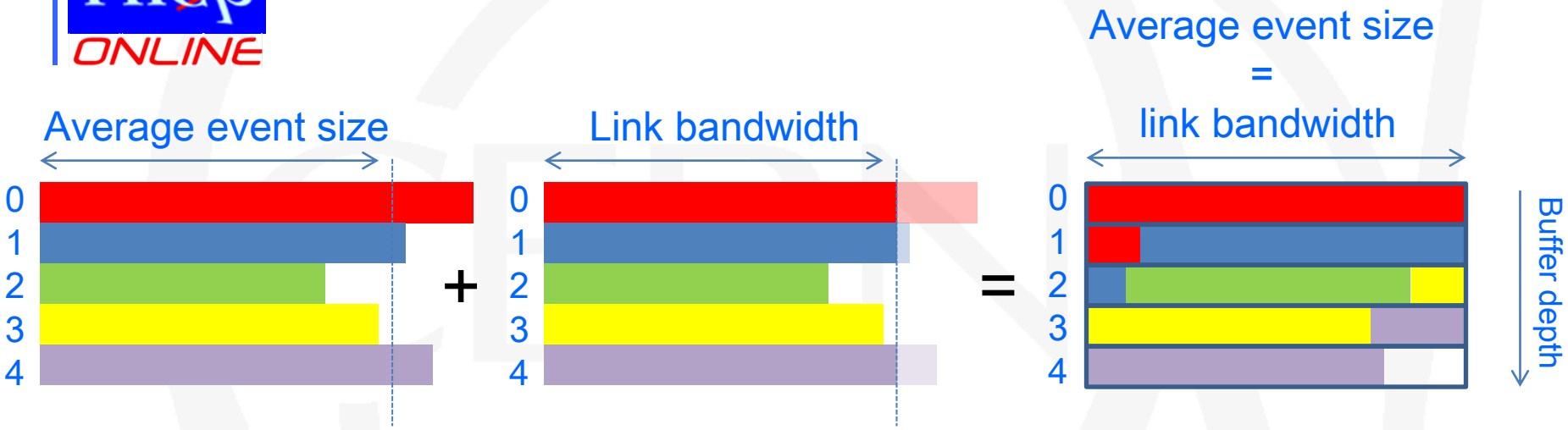
Efficiently use data link bandwidth

- pack data on data link continuously with elastic buffer
- extensive use of GBT (robust FEC vs WideBus mode)
  - ✓ evaluate choices based on complexity vs robustness

Reminder: generic FE data flow scheme

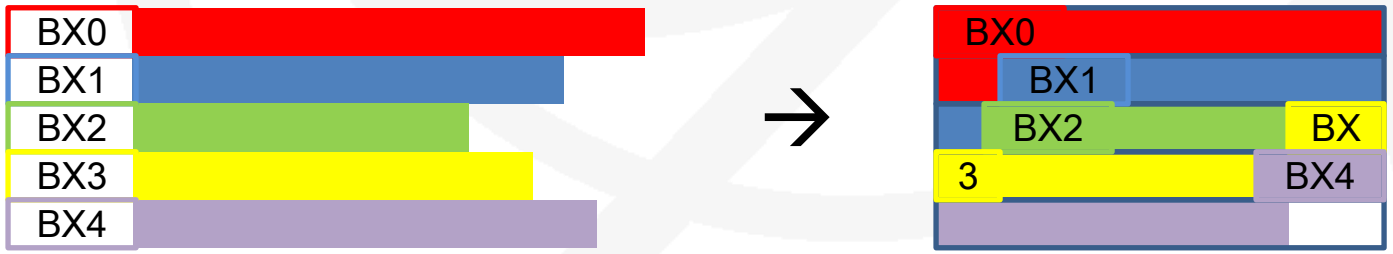


Variable frame length packing algorithm



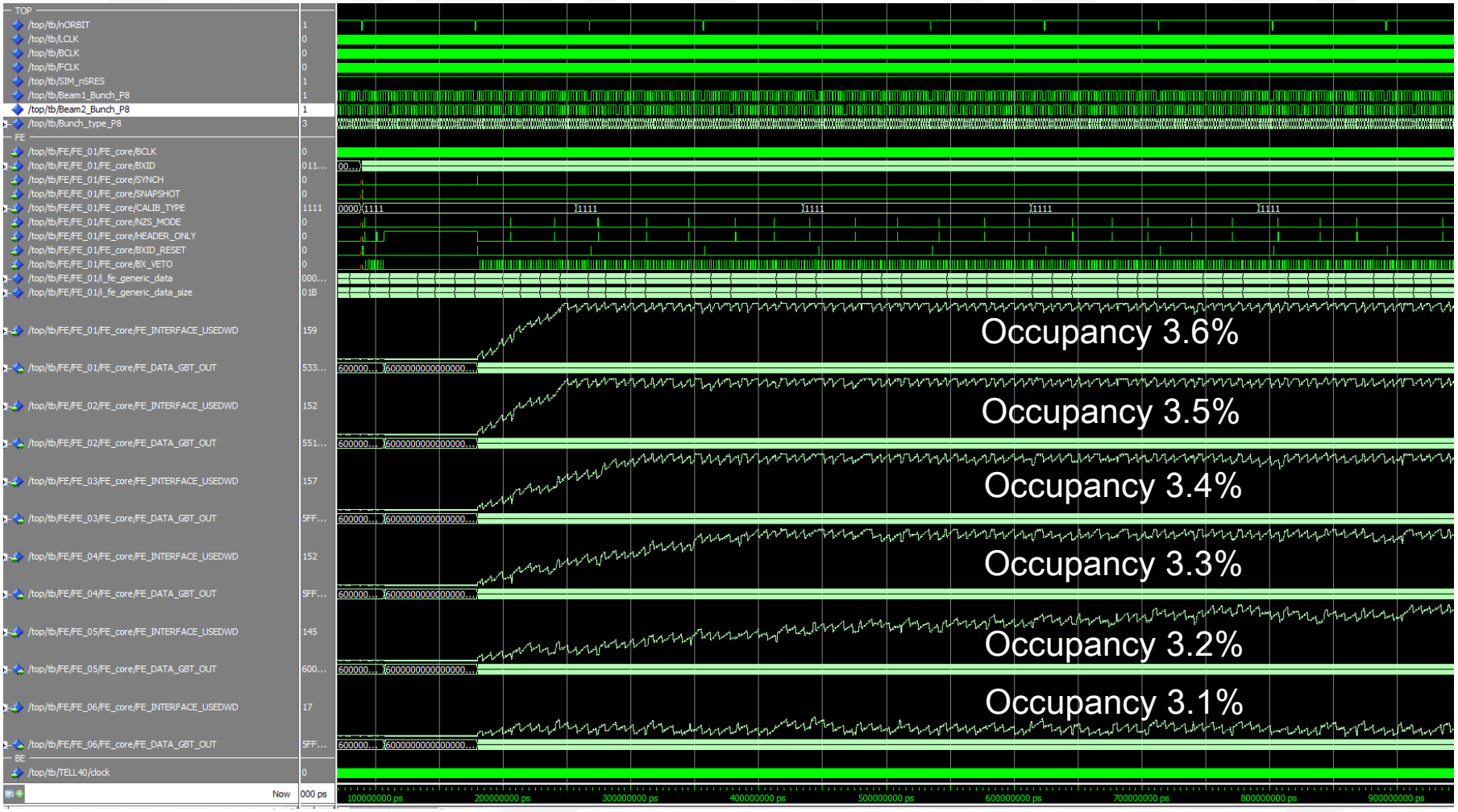
Asynchronous readout: **header is the unique identifier for each event in frame:**

- ✓ Compulsory (tag for each crossing), partly programmable (must contain length of frame+BXID+info)
- ✓ Difficult buffer management, but almost no truncation.
- ✓ Flexible against occupancy fluctuation. Flexible usage of NZS data.
- ✓ Maximum exploitation of bandwidth → reduce # of links.
- ✓ Readout Board uses Header info to decode and separate frames → lots of resources.



Dynamic packing algorithm

This is how the FE buffer would behave in this scenario (example with 500chx4bits + 12bits BXID + 1 «no data» bit BX VETO enabled for all empty-empty)



Fixed vs variable length header in *variable frame length packing*

Variable packing with fixed length header (FV).

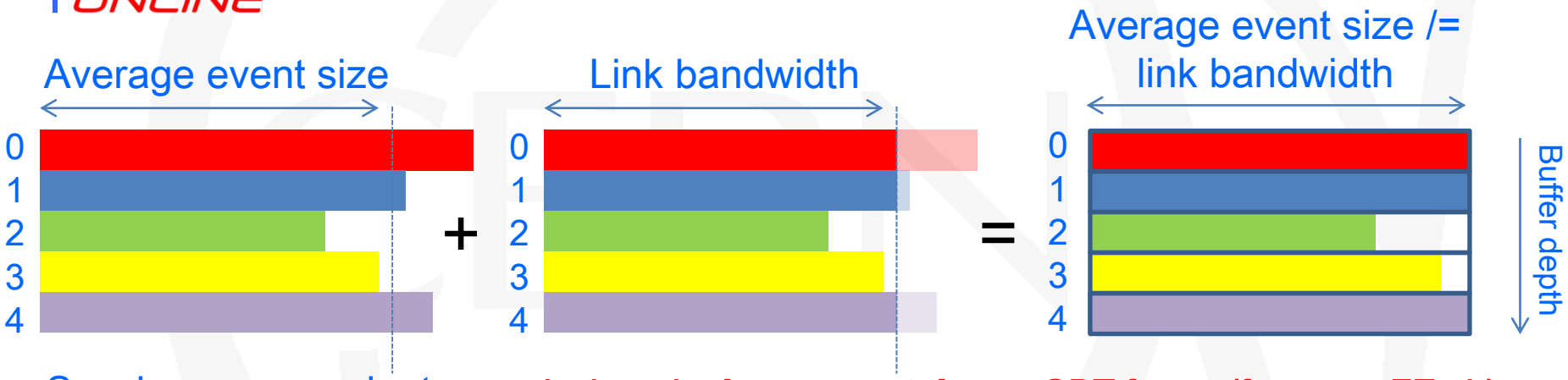
Header field		
BXID	**information**	**Length**
BXID [11:0]	X bits	Y bits

Use case of this encoding is if FE occupancy is very low and want to save on # of links: less bits when no data is sent

Variable packing with variable length header (VV) (fully flexible!).

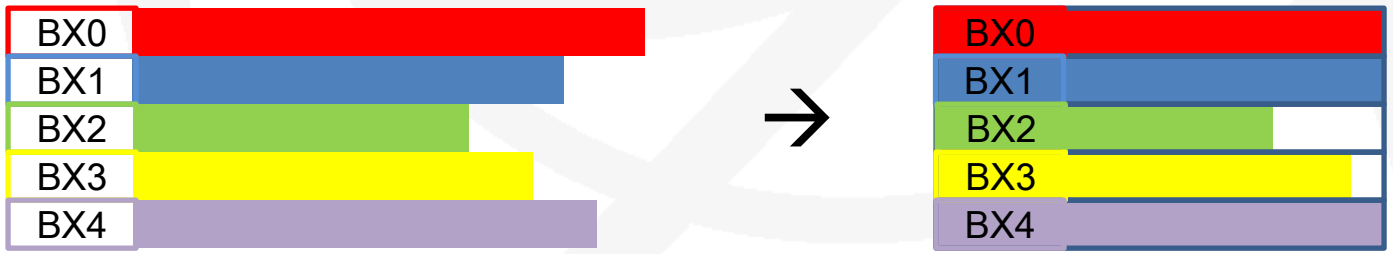
Synch	Header Only or BX Veto or BufferFull	NZS	Output of data filter			
			Header field			Data field
			BXID	No data	Length	
1	X	X	BXID [11:0]	X	X	Synch pattern
0	1	X	BXID [n:0]	1	X	No data
0	0	1	BXID [n:0]	0	NZS code (unique)	Uncompressed data
0	0	0	BXID [n:0]	0	Data length [m:0]	Compressed data

Fixed frame length packing algorithm



Synchronous readout: **one clock cycle → one event → one GBT frame (for many FE ch)**

- ✓ Header more flexible: you can add addresses, hitmaps… Always at the same place.
- ✓ Very simple buffer management, but truncation might happen (depends on avg event size)
- ✓ Not flexible against occupancy problem (depends of avg event size).
- ✓ Loses a bit of bandwidth as empty spaces must be padded.
- ✓ Readout Board uses a fixed length to decode frames → fewer resources



Generic FE algorithms

Algorithms are generic and programmable via configuration package:

- ✓ Programmable
  - Number of channel and size of channels
  - Buffer depth
  - GBT width frame (80 or 112 bits)
  - Header fields
  - Introduce bugs in a controlled way
    - skip BXID, swap BXID etc…
- ✓ Synthesizable
  - Estimate resources in FE (and TELL40…)

Can emulate ANY combination of the FE packing algorithms,
but must be compatible with TELL40 decoding…

Configuration package features III

```
-------------- Incoming FE data ------------------------------------------------------------------------------------
constant data_format_type     : std_logic_vector(2 downto 0) := "001";
-- Select the data format expected :
        -- 001 for variable header length with dynamic data structure
        -- 010 for fixed header length with dynamic data structure
        -- 100 for fixed header length with fixed data structure

constant simulation_fe_data_origin     : std_logic_vector (1 downto 0):= "01";
-- Select the type of injection of data :
        -- 00 nothing
        -- 01 data from the generic data generator
        -- 10 data from the txt file
        -- 11 data from the sub-detector data generator

-- FE specific configuration parameters
--
constant channel_size                  : INTEGER := 4;
constant number_of_channel             : INTEGER := 500;
constant FE_word_size                  : INTEGER := (channel_size*number_of_channel);
constant GBT_word_size                 : INTEGER := 80;
constant GBT_header_size               : INTEGER := 4;
constant FE_bclk_bits                  : INTEGER := 12;
constant FE_size_bits                  : INTEGER := 9;
constant use_data_length_field            : INTEGER range 0 to 1 := 1;  -- 0 no data_length, 1 data_length
constant FE_extra_bits                 : INTEGER := 1;
constant FE_header_size                : INTEGER := FE_bclk_bits+FE_extra_bits+(FE_size_bits*use_data_length_field);
```

Everything is explained in the Mini-DAQ handbook document!

Select the type of encoding + specify header and data fields parameters

Configuration package features IV

```
constant FE_interface_fifo_depth      : std_logic_vector(7 downto 0) := X"A0";

constant occupancy_01                 : real := 3.6;
constant occupancy_02                 : real := 3.5;
constant occupancy_03                 : real := 3.4;
constant occupancy_04                 : real := 3.3;
constant occupancy_05                 : real := 3.2;
constant occupancy_06                 : real := 3.1;

constant NZS_data_length              : INTEGER := channel_size*number_of_channel;

constant FE_BXID_offset               : unsigned(11 downto 0) := X"000";
constant SOL40_TO_FE_TFC_CMD_offset   : unsigned(11 downto 0) := X"D8B";

constant Synch_Pattern_Frame_size     : INTEGER := 10;
constant SYNCH_PATTERN_frame          : std_logic_vector(9 downto 0) := "1011010011";

constant active_fiber                 : std_logic_vector(31 downto 0) := x"0000003F";
```

Change the buffer depth, occupancy for different channels, alignment settings, pattern frame (remember it's programmable)…

Configuration package features V

```
-- Introduce voluntary BXID bugs -- to add to document
-- NOTE 1: if both skip and swap are enabled, skip has priority
-- NOTE 2: if skip_interval and swap_interval have the same value, skip has priority
constant skip_BXID                    : std_logic := '0';
constant skip_BXID_interval           : std_logic_vector(15 downto 0) := X"0545";
constant swap_BXID                    : std_logic := '0';
constant swap_BXID_interval           : std_logic_vector(15 downto 0) := X"0641";
constant skip_BXID_jump               : std_logic_vector(11 downto 0) := X"00C";
-- constant in case of many FE
constant skip_BXID_ApplyToAll         : std_logic := '0';
constant swap_BXID_ApplyToAll         : std_logic := '0';
```

Introduce voluntary bugs in FE code

Nota Bene I

The FE encodings shown here are the ONLY ones allowed in the TELL40 decoding block

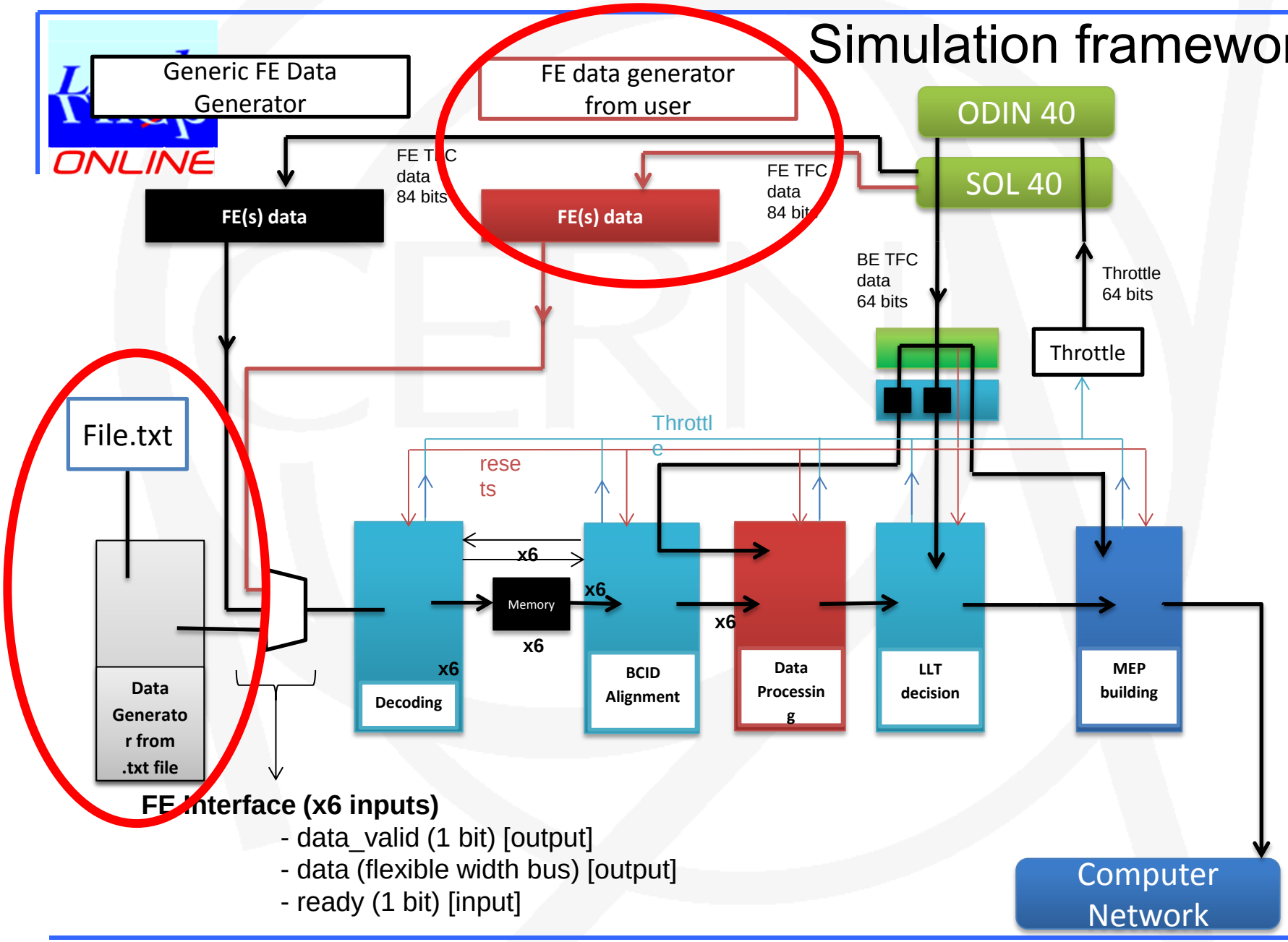
These has been agreed amongst you and if you want to perform a different type of encoding, you should contact us.

There are also other ways to inject FE data to test:

→ From a .txt file

→ From your own HDL code

Simulation framework



FE Interface (x6 inputs)

- data_valid (1 bit) [output]
- data (flexible width bus) [output]
- ready (1 bit) [input]

Your FE code

Only specs:

→ FE data from a .txt file:

[112 or 80 bits data][1 bit data valid]

data valid = 1 == GBT data frame

data valid = 0 == GBT idle frame

→ FE data from your own code:

follow the allowed types of encoding

Everything is explained in the Mini-DAQ handbook document!

Nota Bene II

We expect you to develop your code (eventually):

- Use our configuration package's constant declaration
  - In that way the entire simulation will be set up for you
- Select the type of decoding and see if it works
  - There is a generic wave.do with the signals you are supposed to look at to figure out if it works or not

→ If it doesn't, track a bug (and contact us)

https://lbredmine.cern.ch/projects/amc40/issues/new

Outlook

Next steps:

✓ FE code: Done! If you need help just ask.

✓ TFC code: v0 is out there.

- Will add more features to SODIN with time
  → Ask if you need to enable some features

- Will work more on developing the SOL40 ECS code to FE
  → Help from CBPF to develop an emulation of the GBT-SCA
  → Collaboration with you and ESE group is fundamental (to say the least…)

Conclusion

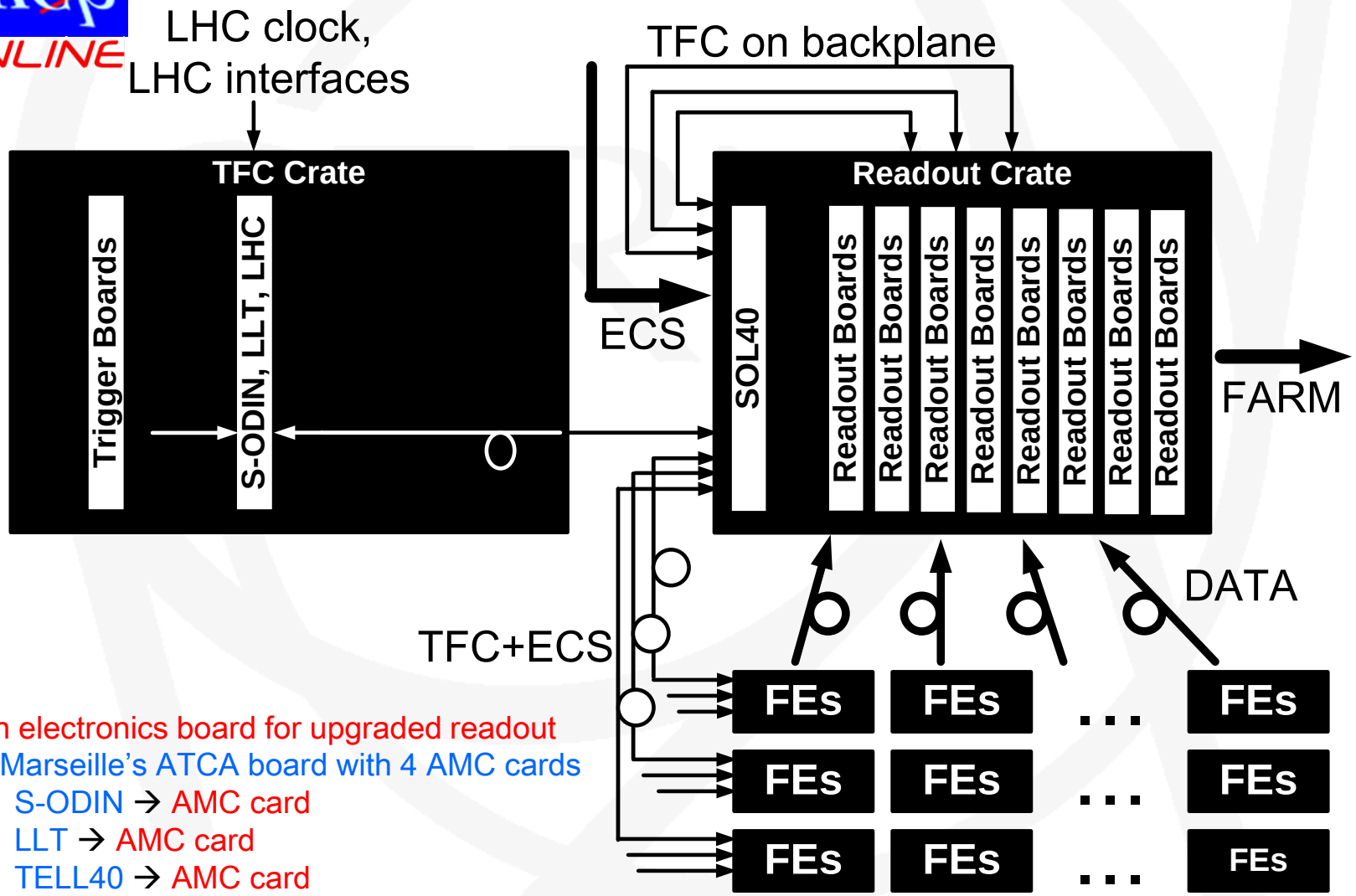
The simulation framework will be our tool to develop hardware code for the upgrade:

→ Please use it, mis-use it and especially, **contribute to it!** We need all the expertise you can possibly provide.

(live) DEMOs

Qs & As?

The upgraded physical readout slice



Common electronics board for upgraded readout system: Marseille's ATCA board with 4 AMC cards

- S-ODIN → AMC card
- LLT → AMC card
- TELL40 → AMC card
- LHC Interfaces → specific AMC card

Latest S-TFC protocol to TELL40

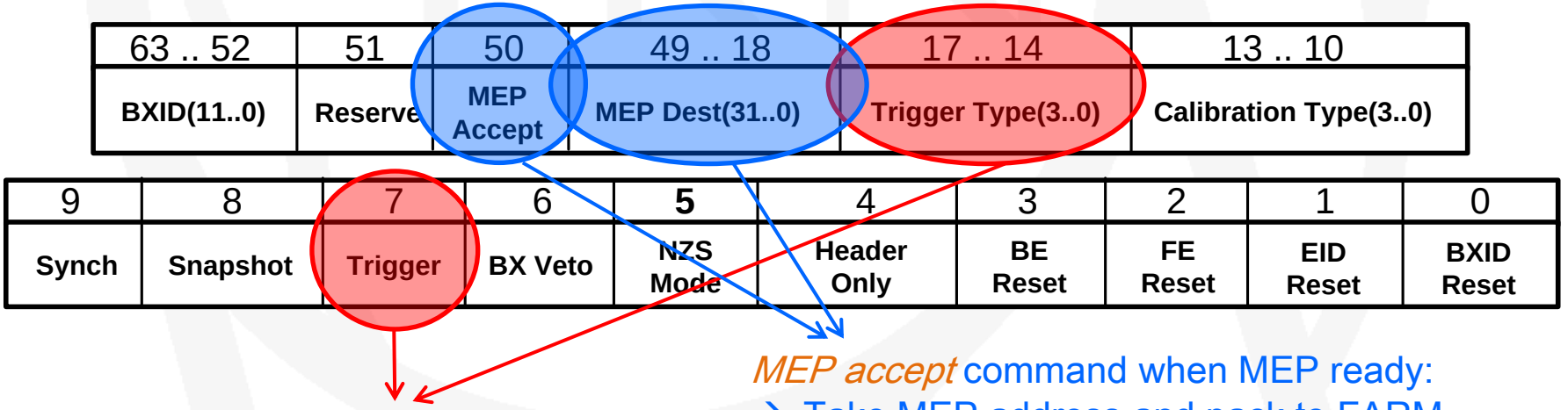
We will provide the TFC decoding block for the TELL40: VHDL entity with inputs/outputs

✓ «Extended» TFC word to TELL40 via SOL40:

→ 64 bits sent every 40 MHz = 2.56 Gb/s (on backplane)

→ packed with 8b/10b protocol (i.e. total of 80 bits)

→ no dedicated GBT buffer, use ALTERA GX simple 8b/10b encoder/decoder



MEP accept command when MEP ready:

→ Take MEP address and pack to FARM

→ No need for special address, dynamic

Constant latency after BXID

✓ THROTTLE information from each TELL40 to SOL40:

- no change: 1 bit for each AMC board + BXID for which the throttle was set

→ 16 bits in 8b/10b encoder

→ same GX buffer as before (as same decoder!)

S-TFC protocol to FE, no change

- ✓ TFC word on downlink to FE via SOL40 embedded in GBT word:
  - → 24 bits in each GBT frame every 40 MHz = 0.98 Gb/s
  - → all commands associated to BXID in TFC word

23 .. 12	11	10	9	8 .. 5	4	3	2	1	0
BXID(11..0)	**Reserve**	**Synch**	**Snapshot**	**Calibration Type(3..0)**	**BX Veto**	**NZS Mode**	**Header Only**	**FE Reset**	**BXID Reset**

Put local configurable delays for each TFC command

- GBT does not support individual delays for each line
- Need for «local» pipelining: detector delays+cables+operational logic (i.e. laser pulse?)
  - → DATA SHOULD BE TAGGED WITH THE CROSSING TO WHICH IT BELONGS!

TFC word will arrive before the actual event takes place

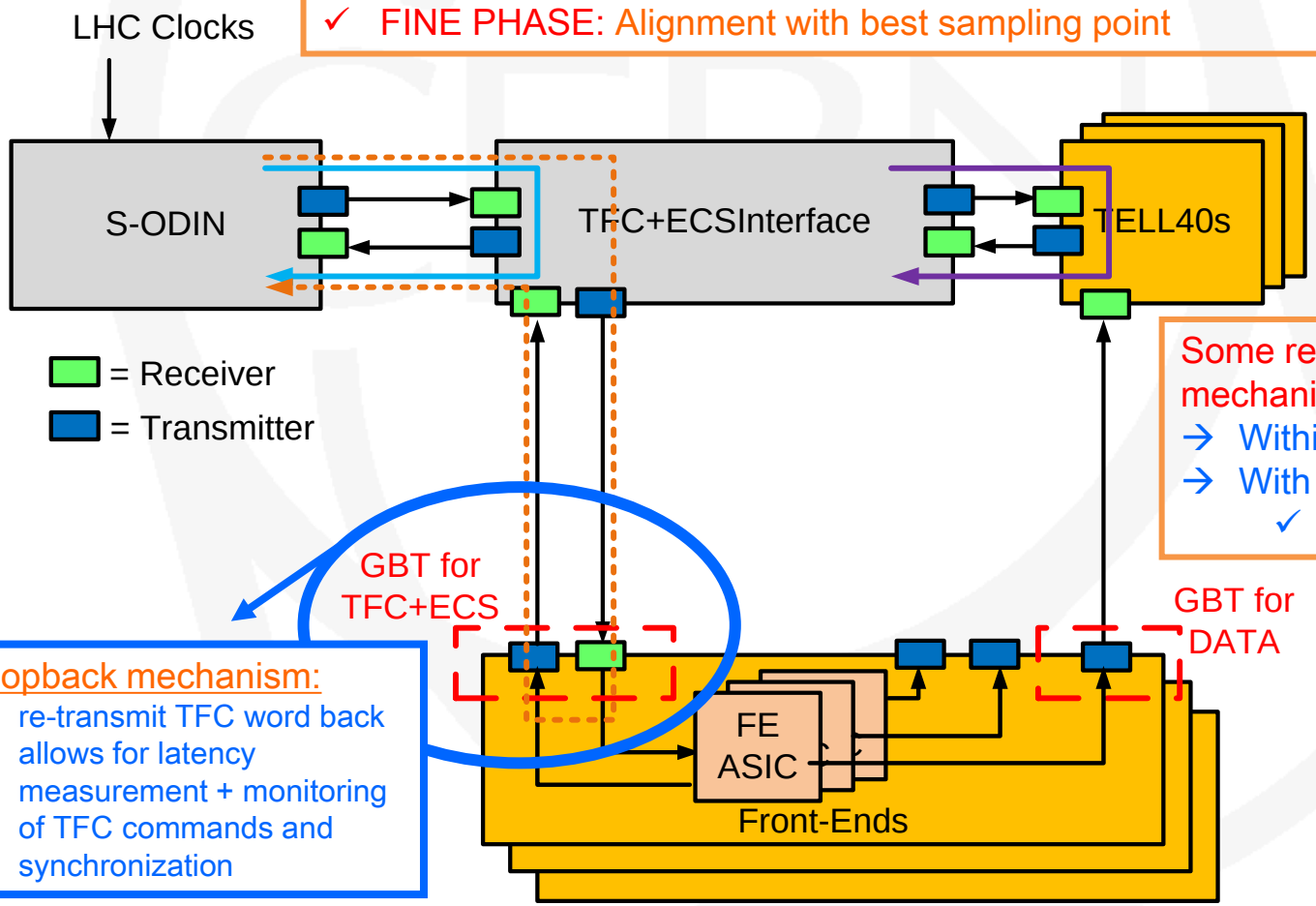
- To allow use of commands/resets for particular BXID
- Accounting of delays in S-ODIN: for now, 16 clock cycles earlier + time to receive
- Aligned to the furthest FE (simulation, then in situ calibration!)

TFC protocol to FE has implications on GBT configuration and ECS to/from FE

- see specs document!

Timing distribution

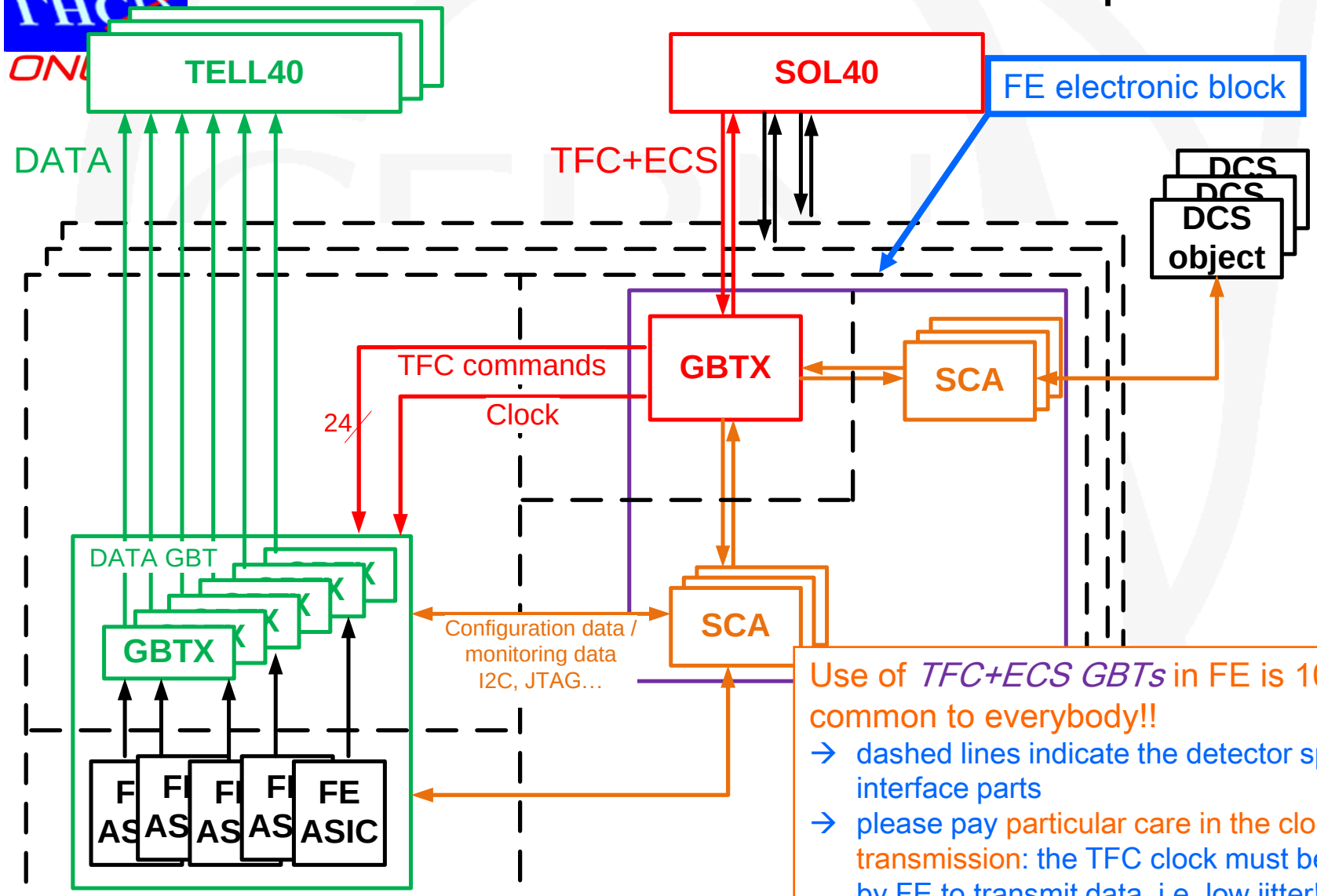
From TFC point of view, we ensure constant:
- ✓ LATENCY: Alignment with BXID
- ✓ FINE PHASE: Alignment with best sampling point



Some resynchronization mechanisms envisaged:
- → Within TFC boards
- → With GBT
  - ✓ No impact on FE itself

Loopback mechanism:
- → re-transmit TFC word back
- → allows for latency measurement + monitoring of TFC commands and synchronization

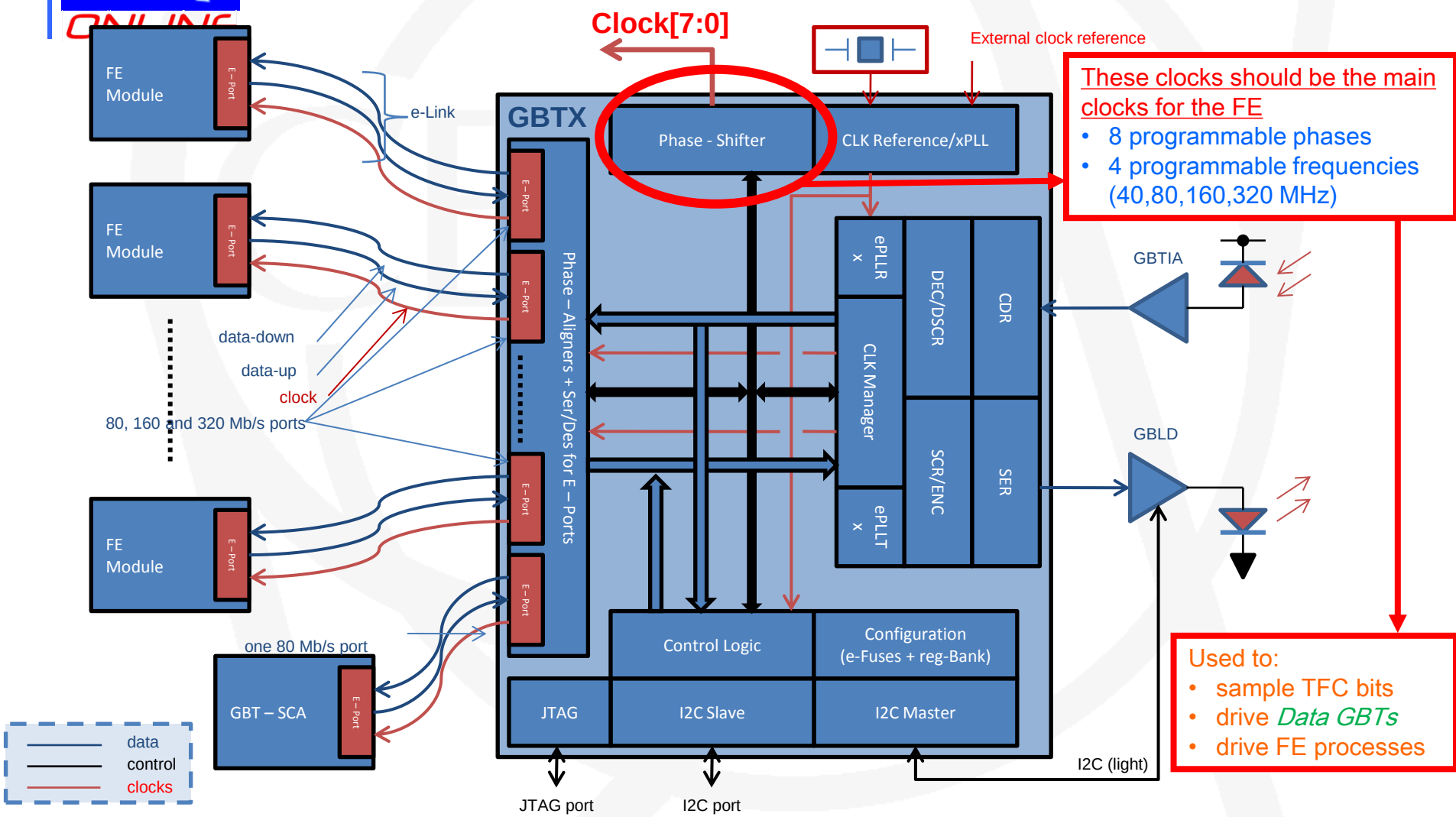
How to decode TFC in FE chips?



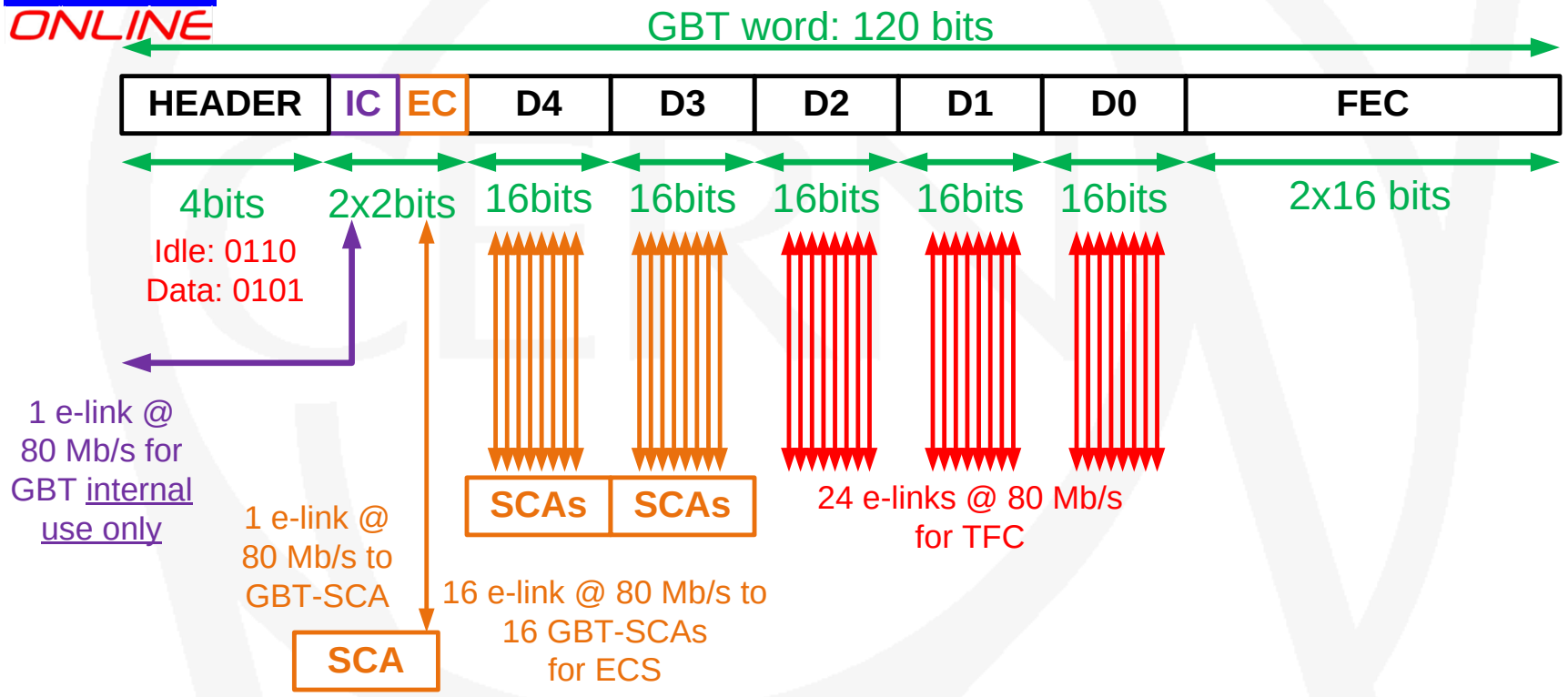
Use of *TFC+ECS GBTs* in FE is 100% common to everybody!!

- → dashed lines indicate the detector specific interface parts
- → please pay particular care in the clock transmission: the TFC clock must be used by FE to transmit data, i.e. low jitter!
  - ✓ Kapton cable, crate, copper between FE ASICs and GBTX

The TFC+ECS GBT



The TFC+ECS GBT protocol to FE

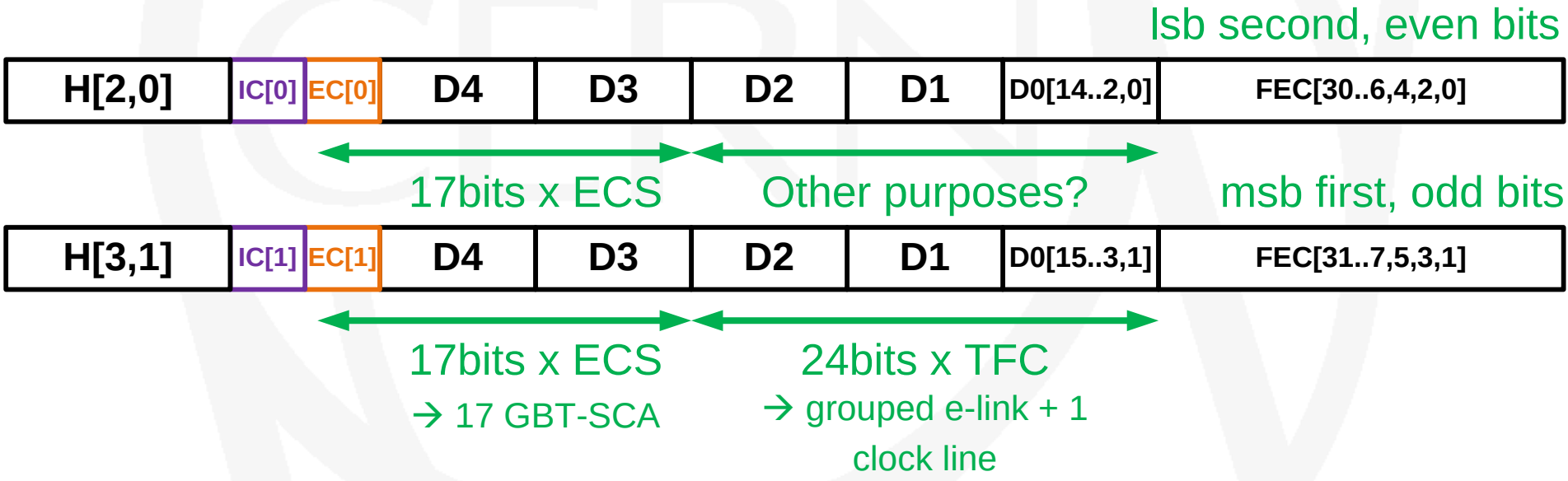


→ TFC protocol has direct implications in the way in which GBT should be used everywhere

- 24 e-links @ 80 Mb/s dedicated to TFC word:
  - ✓ use 80 MHz phase shifter clock to sample TFC parallel word
- TFC bits are packed in GBT frame so that they all come out on the same clock edge
  - ✓ We can repeat the TFC bits also on consecutive 80 MHz clock edge if needed

→ Leftover 17 e-links dedicated to GBT-SCAs for ECS configuring and monitoring (see later)

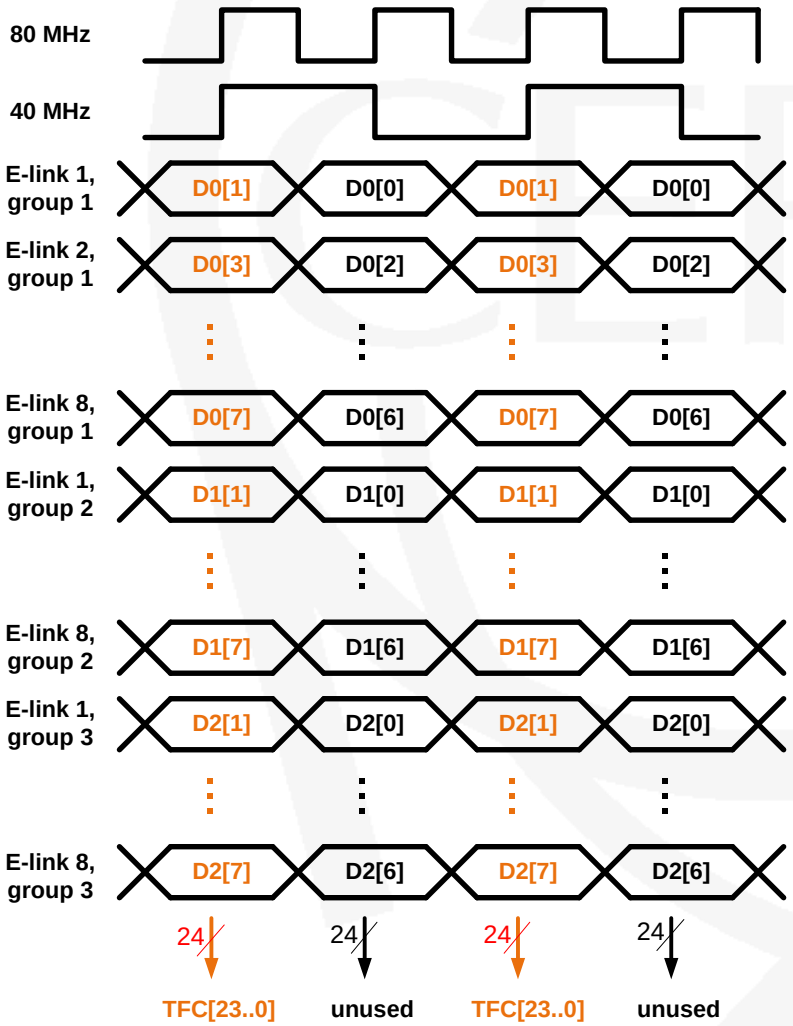
Words come out from GBT at 80 Mb/s



In simple words:

- Odd bits of GBT protocol on rising edge of 40 MHz clock (first, msb),
- Even bits of GBT protocol on falling edge of 40 MHz clock (second, lsb)

TFC decoding at FE after GBT



This is crucial!!

→ we can already specify where each TFC bit will come out on the GBT chip

→ this is the only way in which FE designers still have minimal freedom with GBT chip

- ✓ if TFC info was packed to come out on only 12 e-links (first odd then even), then decoding in FE ASIC would be mandatory!
- ✓ which would mean that the GBT bus would have to go to each FE ASIC for decoding of TFC command

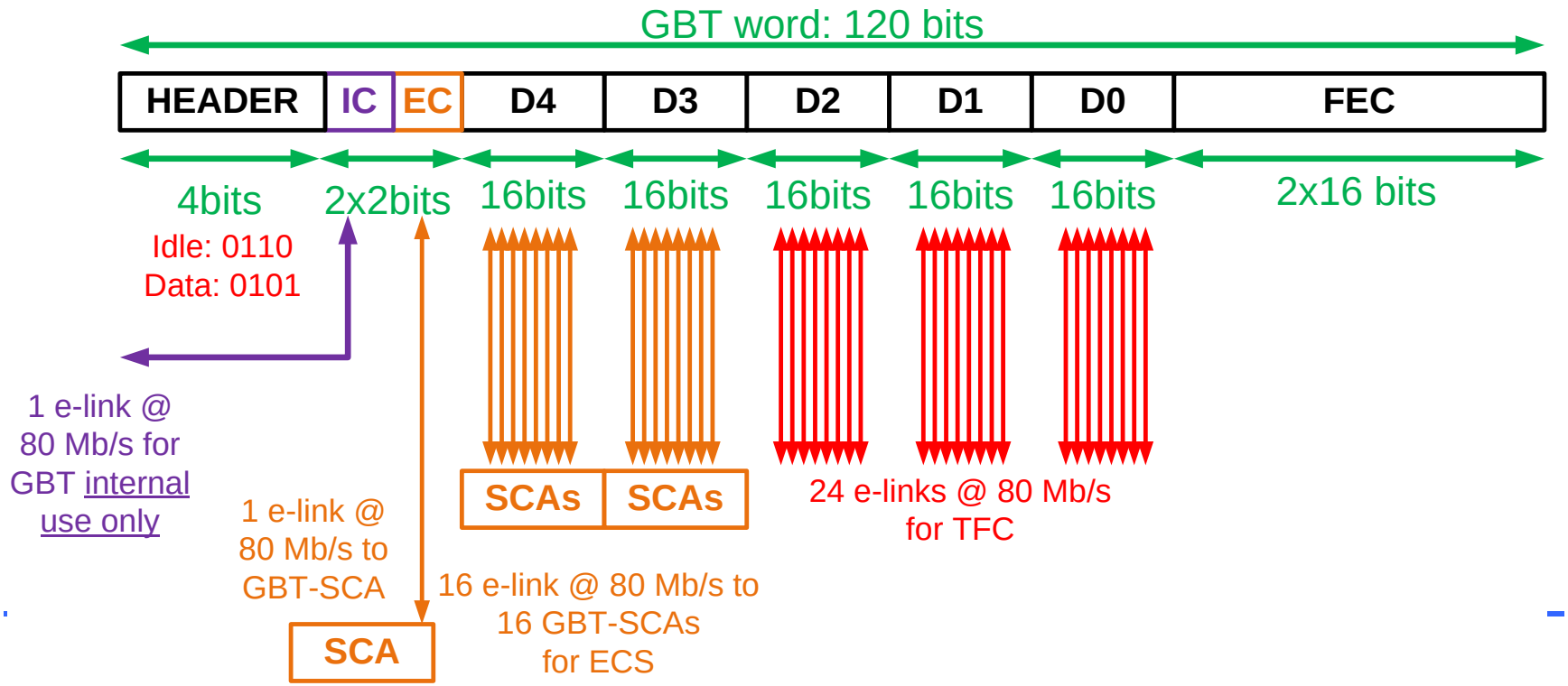
→ there is also the idea to repeat the TFC bits on even and odd bits in TFC protocol

- ✓ would that help?
- ✓ FE could tie logical blocks directly on GBT pins…

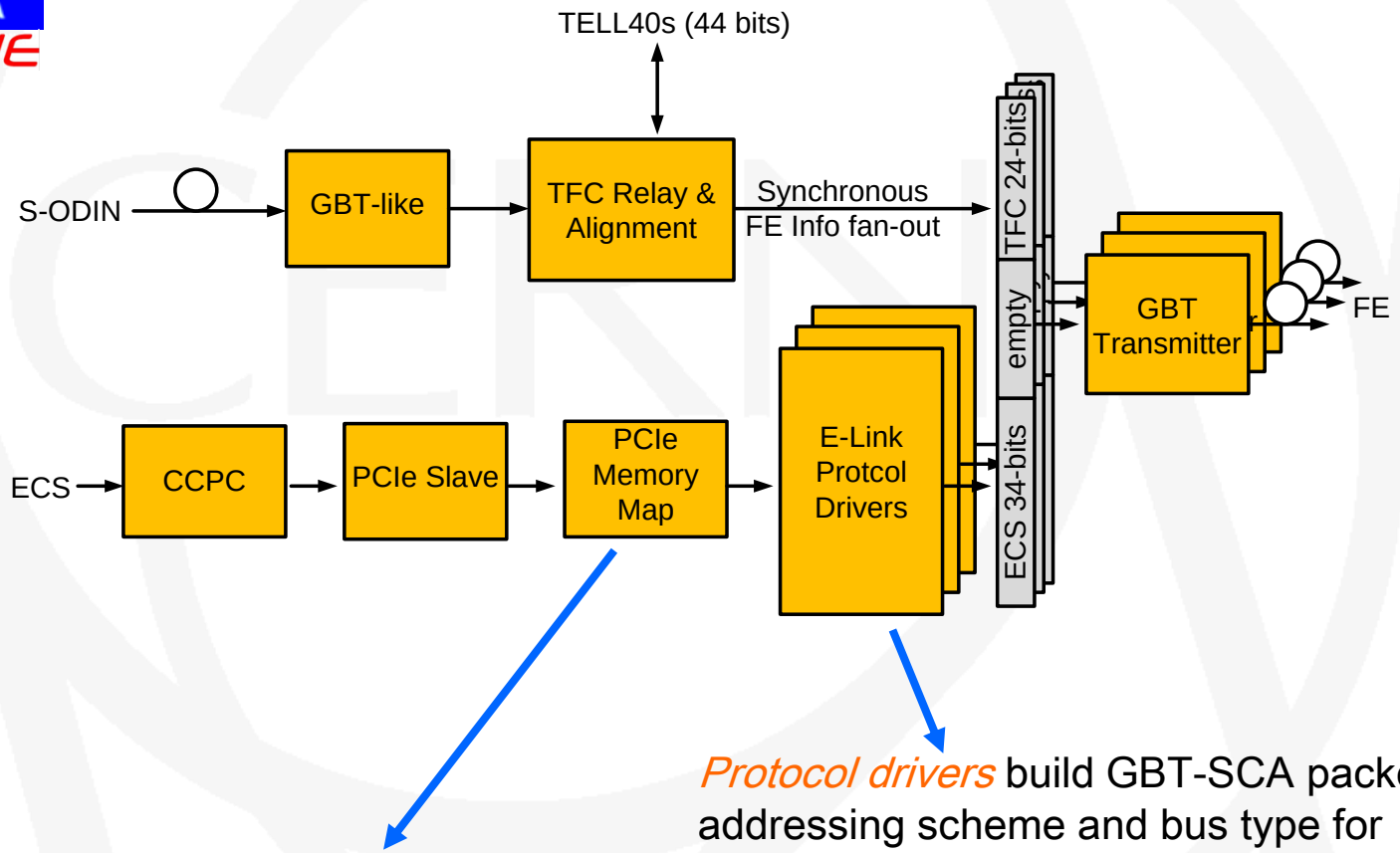
Now, what about the ECS part?

Each pair of bit from ECS field inside GBT can go to a GBT-SCA

- One GBT-SCA is needed to configure the *Data GBTs* (EC one for example?)
- The rest can go to either FE ASICs or DCS objects (temperature, pressure) via other GBT-SCAs
  - ✓ GBT-SCA chip has already everything for us: interfaces, e-links ports ..
    - → No reason to go for something different!
  - ✓ However, «silicon for SCA will come later than silicon for GBTX»…
    - → We need something while we wait for it!



SOL40 encoding block to FE!

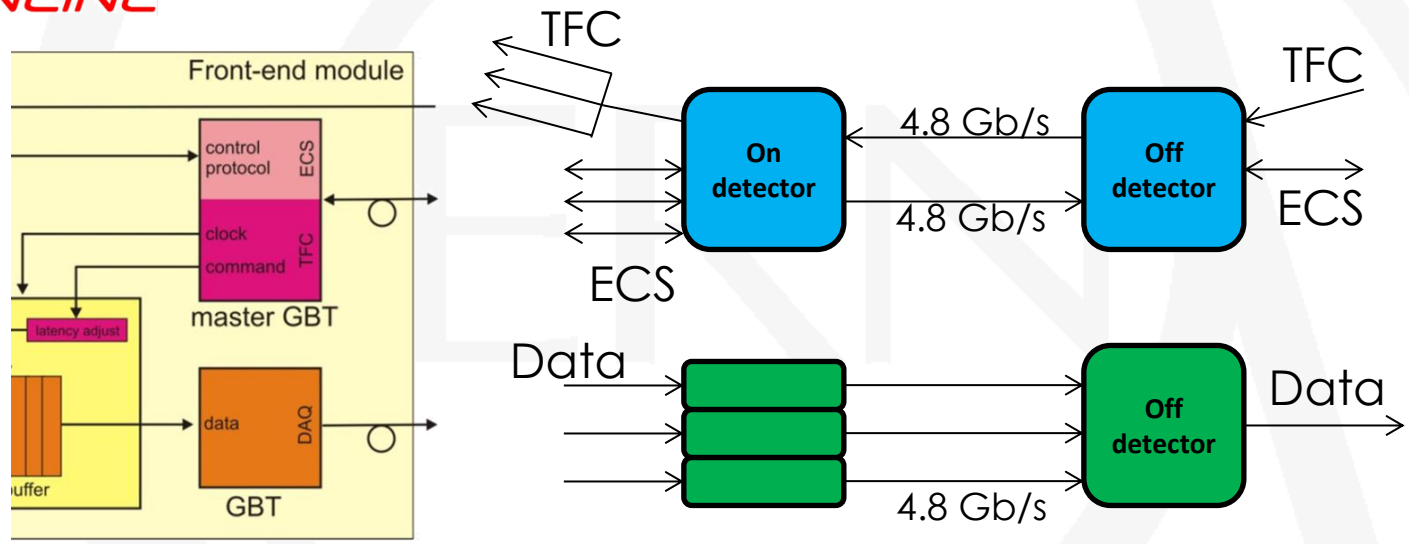


Memory Map with internal addressing scheme for GBT-SCA chips + FE chips addressing, e-link addressing and bus type: *content of memory loaded from ECS*

Protocol drivers build GBT-SCA packets with addressing scheme and bus type for associated GBT-SCA user busses to selected FE chip

→ *Basically each block will build one of the GBT-SCA supported protocols*

Fast & Slow Control to FE



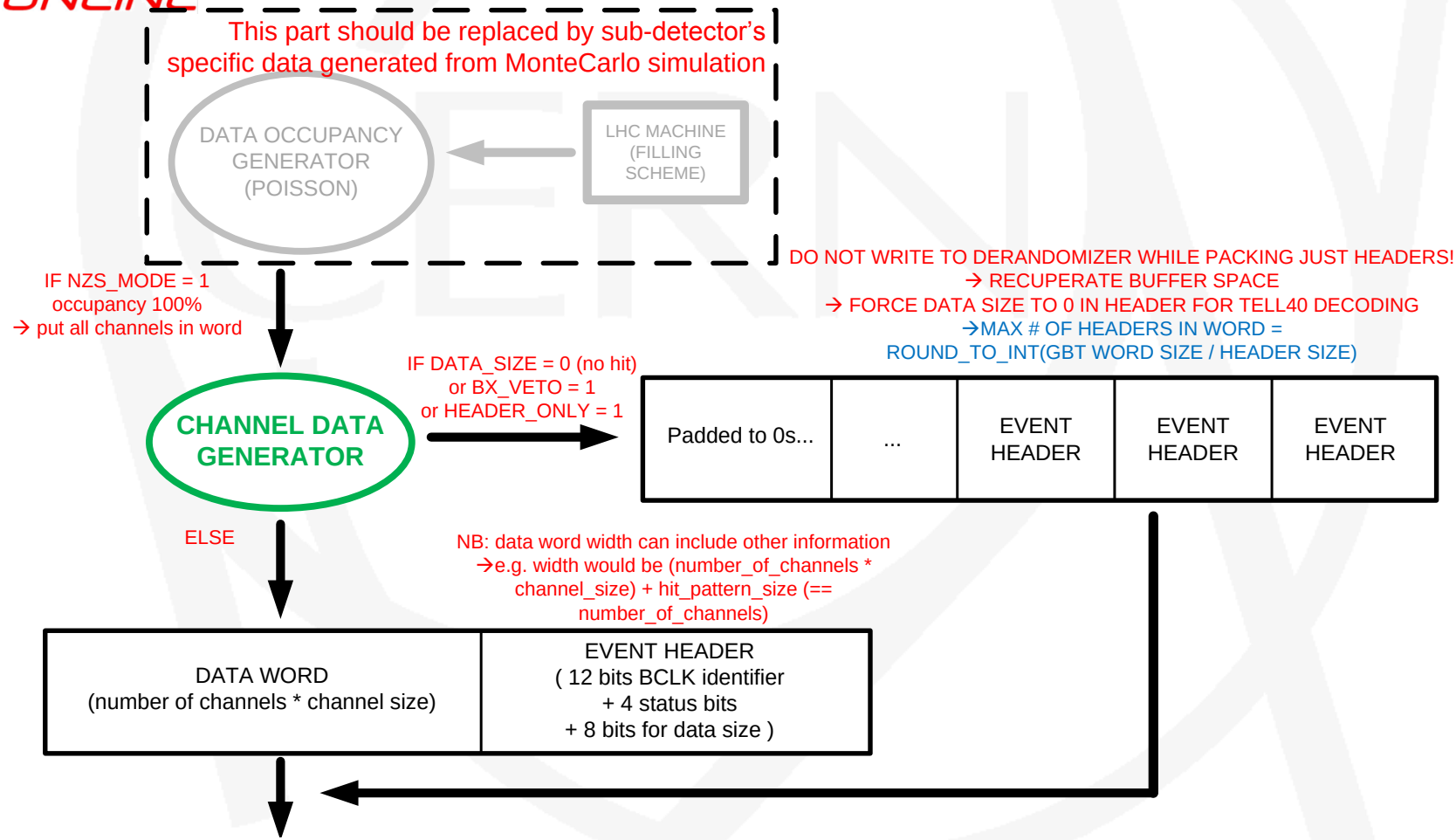
Separate links between controls and data

- A lot of data to collect
- Controls can be fanned-out (especially fast control)

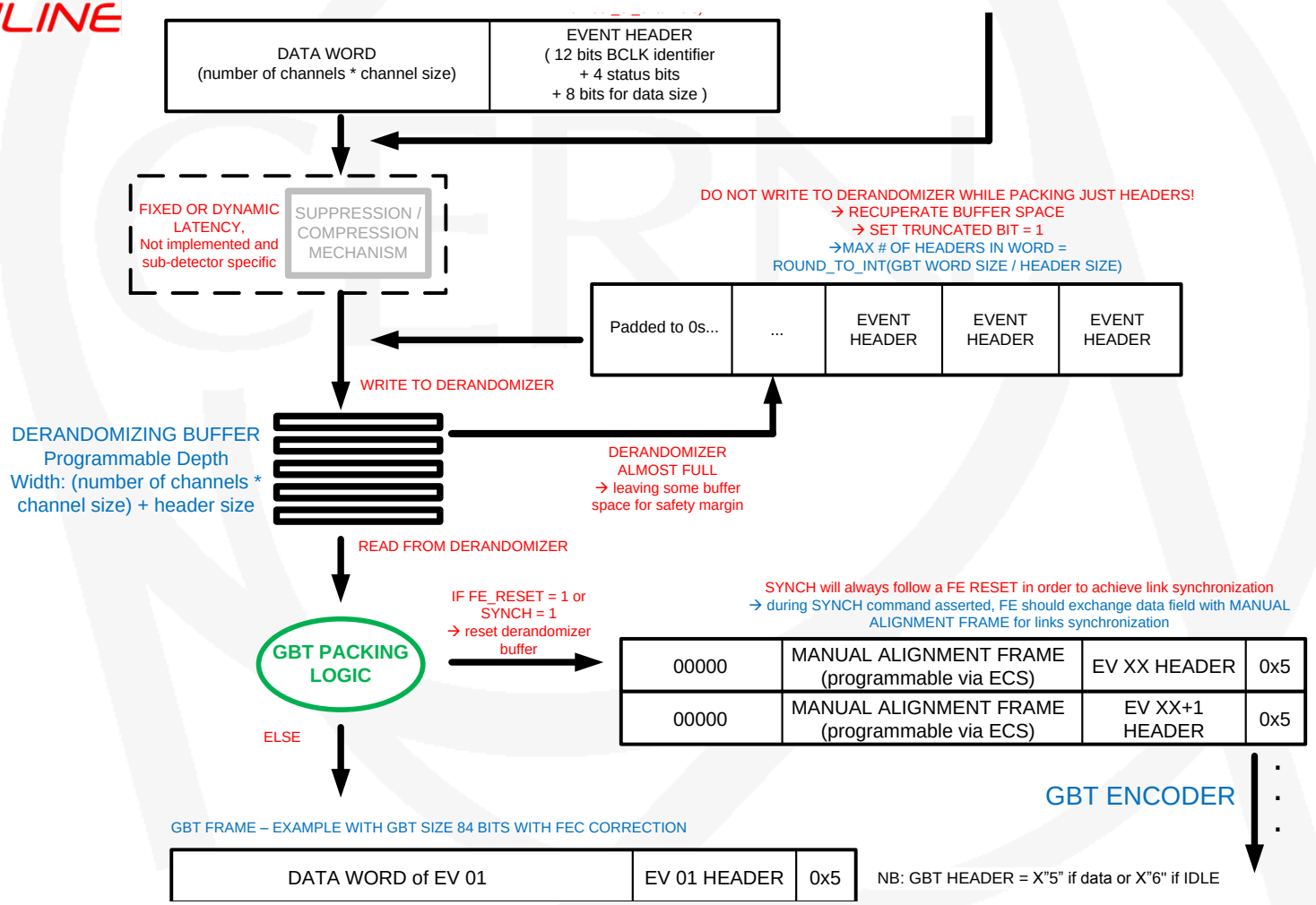
Compact links merging Timing, Fast and Clock (TFC) and Slow Control (ECS).

- Extensive use of GBT as Master GBT to drive Data GBT (especially for clock)
- Extensive use of GBT-SCA for FE configuration and monitoring

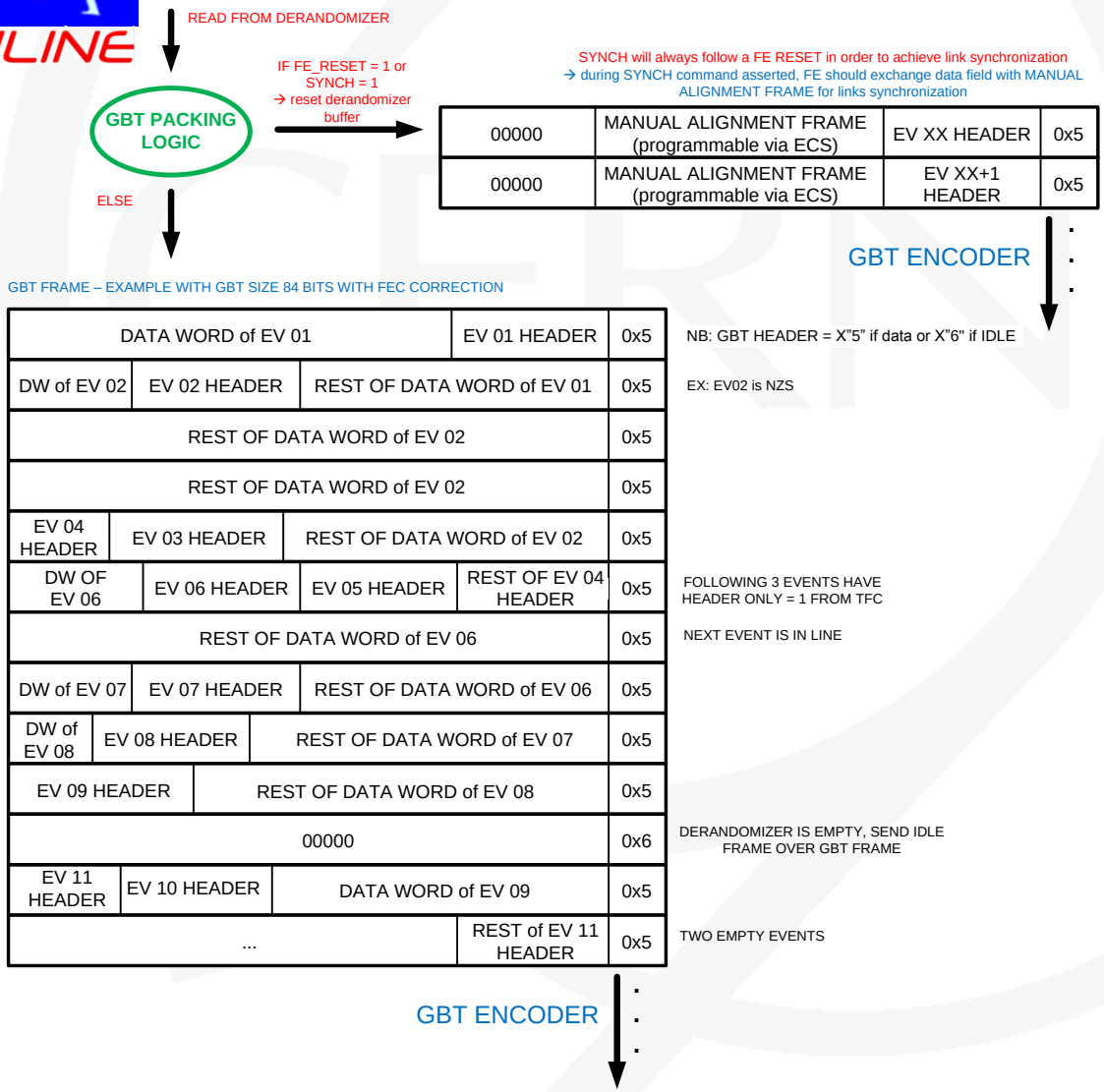
The code: FE data generator



The code: FE buffer manager



The code: GBT dynamic packing

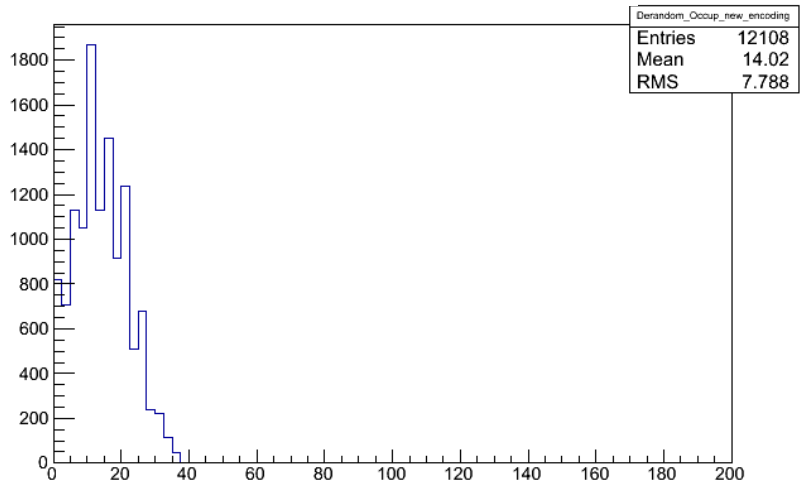


Very important to analyze simulation output bit-by-bit and clock-by-clock!

Studied differences in efficiency

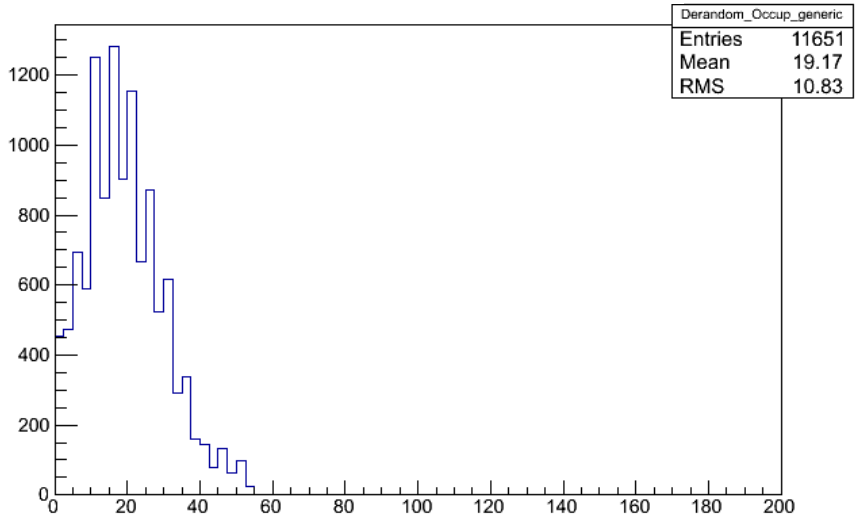
This is the usual example:
500 channels of 4 bits each, occupancy 3.1%, buffer depth 160, 12 bits of BXID

Dynamic with dynamic header



Buffer occupancy over 500 us

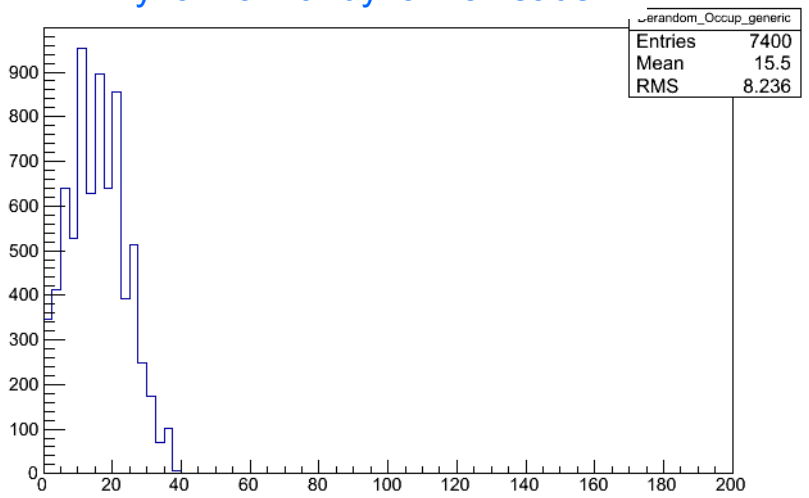
Dynamic with fixed header



Studied differences in efficiency

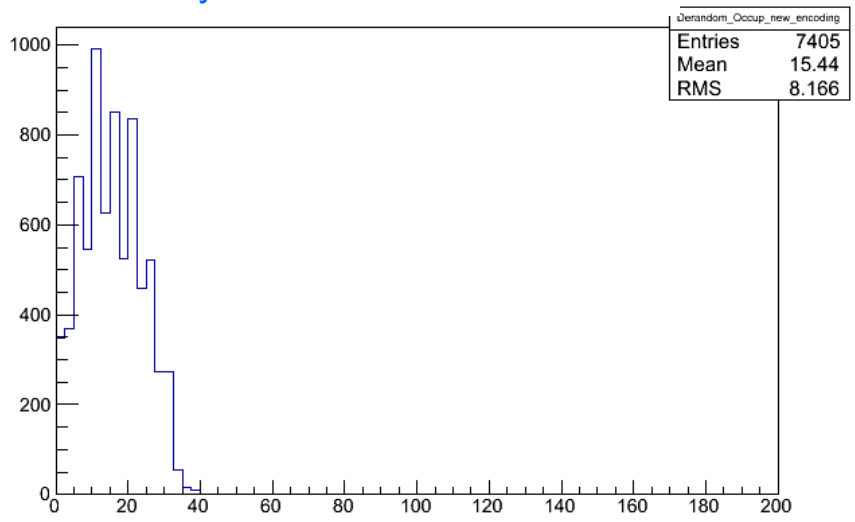
This is just another example:
500 channels of 4 bits each, occupancy 3.6%, buffer depth 160, **4 bits of BXID**

Dynamic with dynamic header

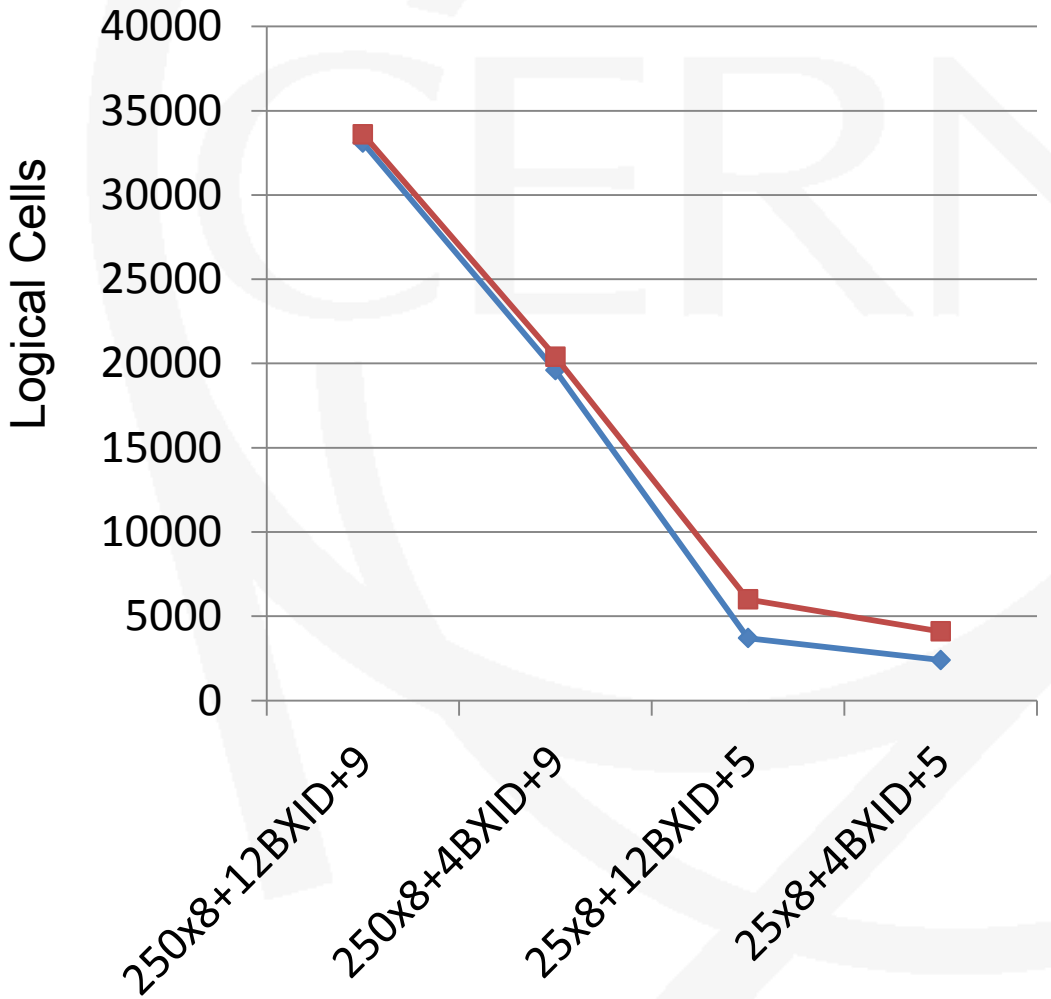


Buffer occupancy over 500 us

Dynamic with fixed header



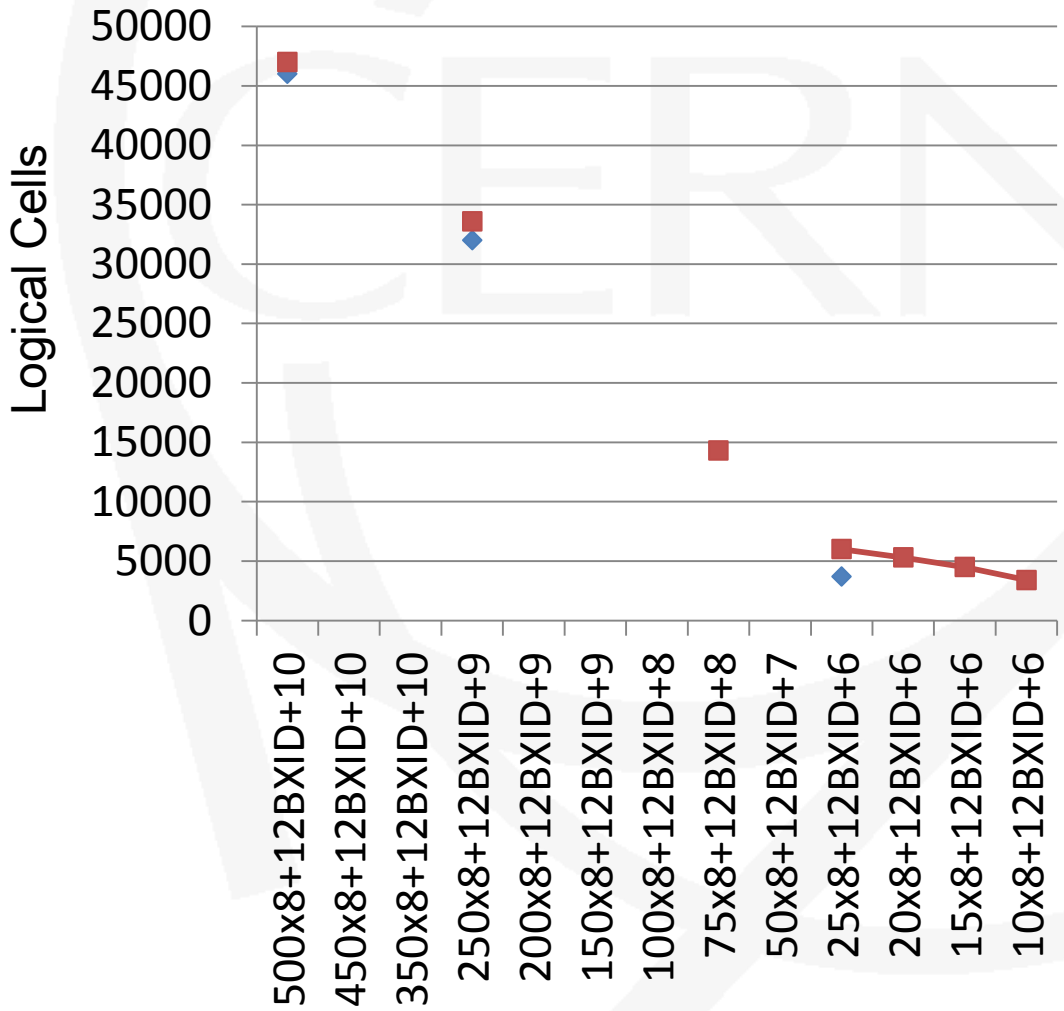
Compared resources needed for different encodings



This is for the ENCODING.
This is per GBT link!

Variable encoding might help you save in fibers, but the cost will rise in FPGA/ASICs resources!

Compared resources needed for different encodings

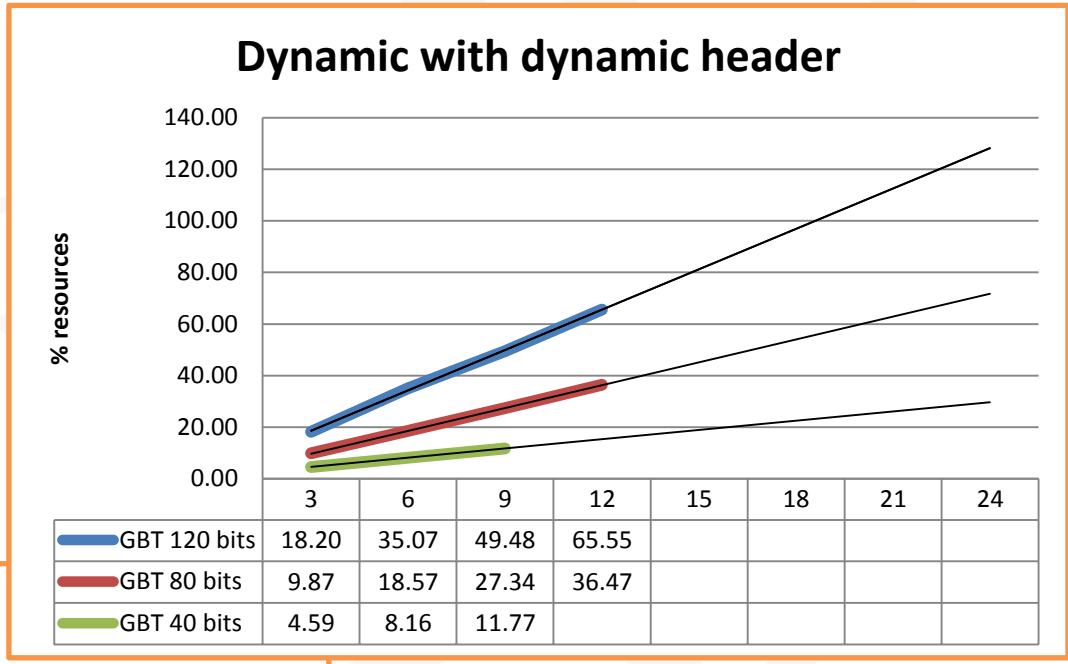


This is for the ENCODING.
This is per GBT link!

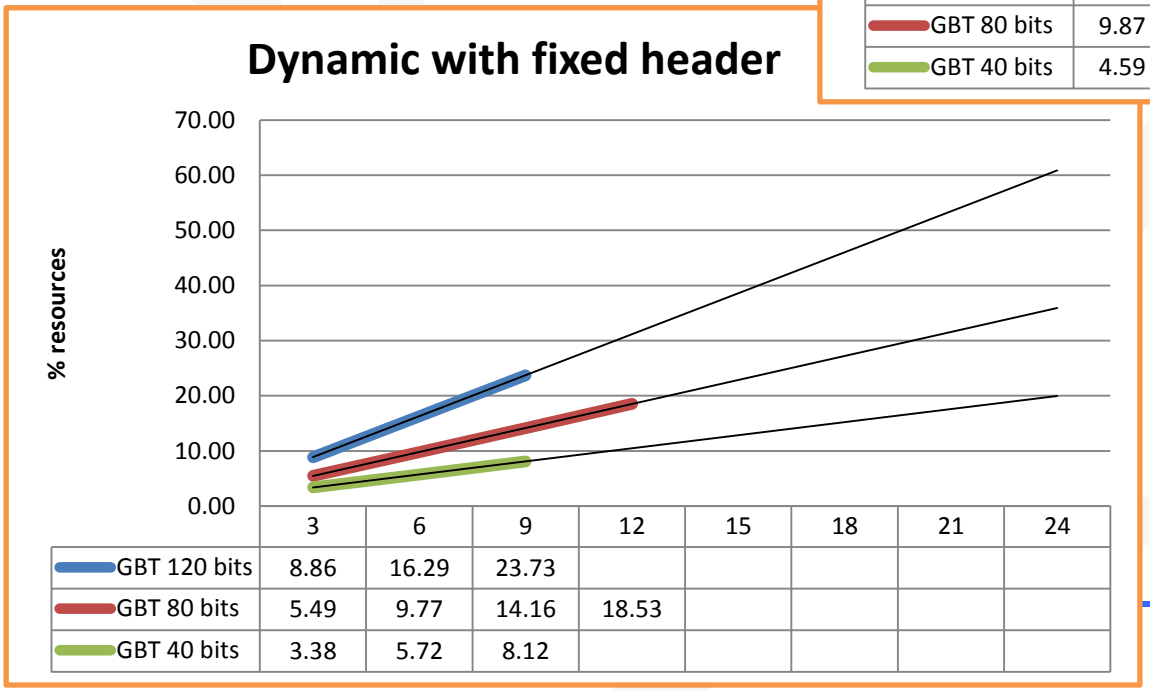
NB: Fixed encoding is 460 LC!
→ 10-100x less
→ CALO & MUON use case - they need fixed latency for the LLT!

Studied impact on TELL40 resources

This is for the DECODER in TELL40.



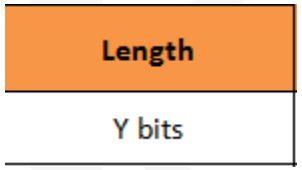
	3	6	9	12	15	18	21	24
GBT 120 bits	18.20	35.07	49.48	65.55				
GBT 80 bits	9.87	18.57	27.34	36.47				
GBT 40 bits	4.59	8.16	11.77					



	3	6	9	12	15	18	21	24
GBT 120 bits	8.86	16.29	23.73					
GBT 80 bits	5.49	9.77	14.16	18.53				
GBT 40 bits	3.38	5.72	8.12					

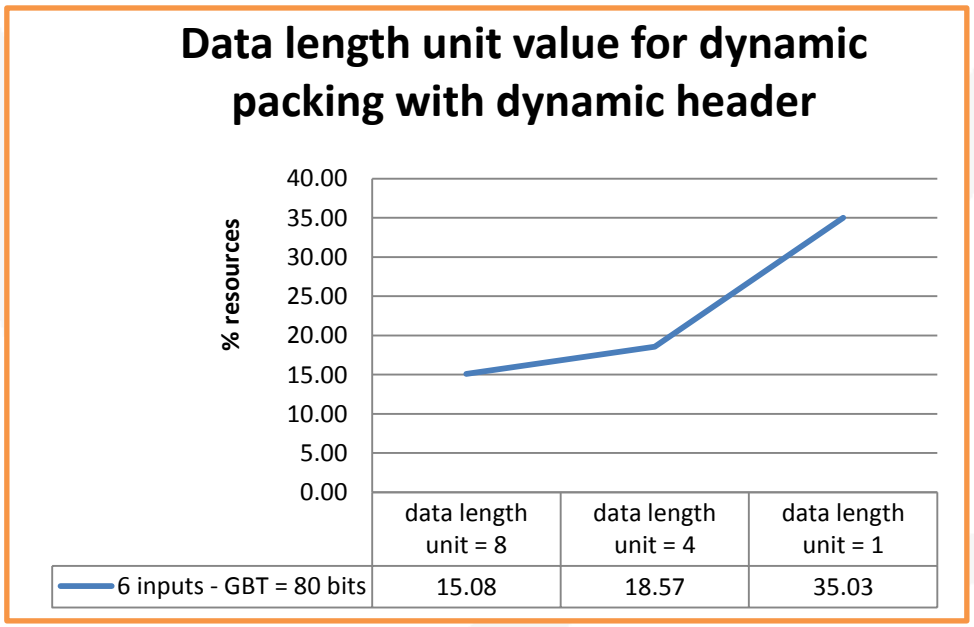
Studied impact on TELL40 resources

Length field will likely contain the number of channels hit
(not the length of the data word – that would require more bits)
Each channel has a "data length unit value" (i.e. size of each channel)



Ex: Length (8 bits) is 0x0A = 10
If data length unit value = 1 : real data length = 10bits
If data length unit value = 4 : real data length = 40bits
If data length unit value = 8 : real data length = 80bits

Test done with dynamic packing with dynamic header



The data length unit value should be bigger or equal to 4.

We should forbid smaller than 4.

The code: configuration

FE generic data generator is fully programmable:

- ✓ Number of channels associated to GBT link
- ✓ Width of each channel
- ✓ Derandomizer depth
- ✓ Mean occupancy of the channels associated to GBT link
- ✓ Size of GBT frame (80 bits or WideBus + GBT header 4 bits)

Extremely flexible and easy to configure with parameters

Covers almost all possibilities (almost…)

- ✓ Including flexible transmission of NZS and ZS

Including TFC commands as defined in specs

- ✓ Study dependency of FE buffer behaviour with TFC commands
- ✓ Study effect of packing algorithm on TELL40
- ✓ Study synchronization mechanism at beginning of run
- ✓ Study re-synchronization mechanism when de-synchronized
- ✓ Etc… etc… etc…

And it is fully synthesizable… ☺

Conclusions

Packing mechanism as specified in our document is feasible.

- ✓ Will be used temporarily to emulate FE generated data in global readout and TFC simulation.

However, very big open questions:

- ✓ Is your FE compatible with such scheme? What about such code in an ASIC?
- ✓ Behaviour of FE derandomizer will strongly depend on your compression or suppression mechanism.
  - If dynamic could create big latencies
  - If your data does not come out of order can become quite complicated…
- ✓ Behaviour of FE derandomizer will strongly depend on TFC commands
  - FE buffer depth should not rely on having a BX VETO! Aim at a bandwidth for fully 40 MHz readout → BX VETO solely to discard events synchronously.
  - What about SYNCH command? When do you think you can apply it? Ideally after derandomizer and after suppression/compression, but…
- ✓ How many clock cycles do you need to recover from an NZS event?
  - Can you handle consecutive NZS events?

Old TTC system support and running two systems in parallel

We already suggested the idea of a *hybrid system*:

reminder: L0 electronics relying on TTC protocol

→ part of the system runs with old TTC system

→ part of the system runs with the new architecture

How?

1. Need connection between S-ODIN and ODIN (bidirectional)
   → use dedicated RTM board on S-ODIN ATCA card
2. In an early commissioning phase ODIN is the master, S-ODIN is the slave
   → S-ODIN task would be to distribute new commands to new FE, to new TELL40s, and run processes in parallel to ODIN
   → ODIN tasks are the ones today + S-ODIN controls the upgraded part
   ✓ In this configuration, upgraded slice will run at 40 MHz, but positive triggers will come only at maximum 1.1MHz…
   - Great testbench for development + tests + apprenticeship…
   - Bi-product: improve LHCb physics programme in 2015-2018…
3. In the final system, S-ODIN is the master, ODIN is the slave
   → ODIN task is only to interface the L0 electronics path to S-ODIN and to provide clock resets on old TTC protocol

Firmware for Mini-DAQ

Compilation of first TFC + TELL40 firmware

Preparation of simulation and compilation framework

Done!

Integrate LLI and DAQ core

Done!

(Basic) software to control Mini-DAQ

Getting done!

Tests & tests & tests
then deploy