RD53B Command & Data Protocol

CERN - March 1st 2019

Roberto Beccherle, INFN - Pisa

Input protocol

Protocol definition: [Downlink is running **@160 Mbit/s**, **clock** and **data** encoded on the same lane]

- All commands are made out of a certain number of **Frames**. Each Frame is a 16-bit string.
- Each **Frame** is made out of two consecutive **Symbols**. Each symbol is an 8-bit string.
- There are 54 possible **Symbols** in the Command grammar.
- One **Frame** is sent every four 40MHz clock or every sixteen 160MHz clock.
- There are two classes of commands:
  - **Fast commands**: They are single frame commands.
    They have **high priority** and can be interleaved inside slow commands.
  - **Slow commands**: They are multiple frame commands. They cannot be interleaved.

Available symbols:

Data00, Data02, Data03, Data04, Data05, Data06, Data07, Data08, Data09, Data10, Data11, Data12, Data13, Data14, Data15, Data16, Data17, Data18, Data19, Data20, Data21, Data22, Data23, Data24, Data25, Data26, Data27, Data28, Data29, Data30, Data31,

Trig_01, Trig_02, Trig_03, Trig_04, Trig_05, Trig_06, Trig_07, Trig_08, Trig_09, Trig_10,

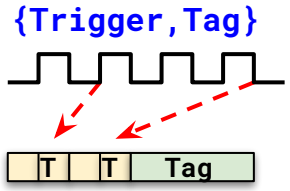
Trig_11, Trig_12, Trig_13, Trig_14, Trig_15,

Cal, **Clear**, GlobalPulse, **PllLock**, RdReg, **ReadTrigger**, WrReg

So there are 32 Data symbols, 15 Trigger symbols and 7 additional ones.

Symbols in **red** are the symbols introduced in the new version of the Command Decoder for RD53B, and where NOT present in RD53A..

Command definition



Sync command:

- **{1000_0001,0111_1110}** Special command, to be sent periodically, to lock the channel. After 64 frames without Sync the channel unlock increments.

Fast commands:

- Clear: **{Clear,Clear}** Replaces old ECR command [Clears whole Data-Path]
- PllLock: **{PllLock, PllLock}** Used to provide a '**10101010**' pattern to ease lock of Pll.
- Trig_NN: **{Trig_NN,Any Symbol}** Triggers data in the Pixel Matrix and tags it with a Tag, a 6 bit number (internally expanded to 8 bit) that depends on the provided Symbol. [**54 * 4 = 216 Tags**]

Slow commands:

- Cal: **{Cal,Cal}, {DataX1,DataX2},{DataX3,DataX4}**
- GlobalPulse: **{GlobalPulse,GlobalPulse} {DataX1,DataX2}**
- RdReg: **{RdReg,RdReg} {DataX1,DataX2},{DataX3,DataX4}**
- ReadTrigger: **{ReadTrigger,ReadTrigger} {DataX1,DataX2}**

DataX1 and **DataX2** determine the **8 bit eTag** that tells the chip what data to read out.

- WrReg **{RdReg,RdReg} {DataX1,DataX2},{DataX3,DataX4} {DataX5,DataX6},[{DataX7,DataX8},{DataX9,DataX10}, {DataX11,DataX12},{DataX13,DataX14},{DataX15,DataX16}]**

Data in **blue** is optional and used just if AutoIncrement mode inside the chip is enabled.

Output from the Chip(s)

Chip Output Data:

- There are 1 to 4 Aurora64/66b @1.28GB/s encoded lanes that form a Simplex Aurora Channel. The Data has strict alignment. [always Same data type on all Lanes]
- Optionally it has one or two 320Mb/s (640Mb/s, if working) Aurora64/66b encoded Lanes that form a Simplex Aurora Channel only used for on chip data merging.

Aurora Channel contents:

Each Aurora Channel might contain Pixel Data, Service Data or No Data.

- **Pixel Data**: Continuous stream of 66 bit Aurora Frames, without separators. Each frame is identified by a '**01**' header followed by 64 bit of data.
- **Service Data**: Single Aurora Frame identified by a '**10**' header followed by 64 bit of data.
- **No Data**: Standard Aurora Idle Frame, also identified by a '**10**' header.

Service Data:

- A single Aurora Frames that contains the value of **two RD53 registers** (Global or Pixel).

Pixel Data:

- All data coming from pixels is **split into 64 bit wide Aurora Frames**.
- Data is generated inside the chip and put in variable length streams.
- Each pixel chip always produces an unknown number of concatenated data streams.
- For each incoming Trigger an outgoing Event will be produced [no event suppression]

Variable length Data Streams (1of2)

Data Stream:

- It is a way to efficiently split Data over 64 bit Aurora frames. **It is NOT an Event!**
- A chip will read out all data coming from the Pixel Region and assemble it in Events.
- Events are of variable length due to Binary Tree data formatting.
- As soon as we have data we will insert it in data streams, and split it on Aurora Frames.

Data Stream Format:

- Each Aurora frame will always use its most significant bit to signal if we are at the beginning of a Stream (**'1'**) of if the data belongs to a previously started stream (**'0'**).
- Therefore in each Aurora Frame we will have 63 bit of Pixel Data.
- A Stream is always started by the 8 bit Tag that identifies the Event currently being processed.
- It is followed by the 6 bit address of the chip column (**ccol**) and the 6 bit of the 8x8 Core address (**crow**).

Core Data:

- Core data is divided in 2x8 regions (two additional bits are used) and encoded using the Binary Tree Encoder.
- A region is encoded into a string that can have from 4 to 30 bits (variable length part).
- The encode address of hit Pixels is followed by one to 16 4 bit ToT fields.

End of Stream:

- After the first data, if we are closer than **E** (EndOfFrame) bits from the end of the Aurora Frame.

Variable length Data Streams (2of2)

End of Stream (cont.):

- If current position from end of Aurora frame is less than E bits we end the stream.
- If not we increment E, by a programmable value, and start adding new data to the stream.
- This programmable value allows to favour longer or shorter Streams.
- New data starts with the next ccol, crow followed by new Core data, or a new Event.
- After next data we check again E and terminate the stream or increment it again.
- As soon as the end condition will be met, so we pad remaining bits of the frame with **'0'**s.
- A Stream is also ended, padding with **'0'**s, if there is no more data to be sent.

Events inside a Stream:

- Streams are built in a way that it is not ensured that Events are fully contained in it.
- When we end an Event we have three possible options:

1. We are at the end of a Stream: In this case we end both the Event and the Stream.
2. We are not at the end of a Stream: We will add data from a new event to the Stream.

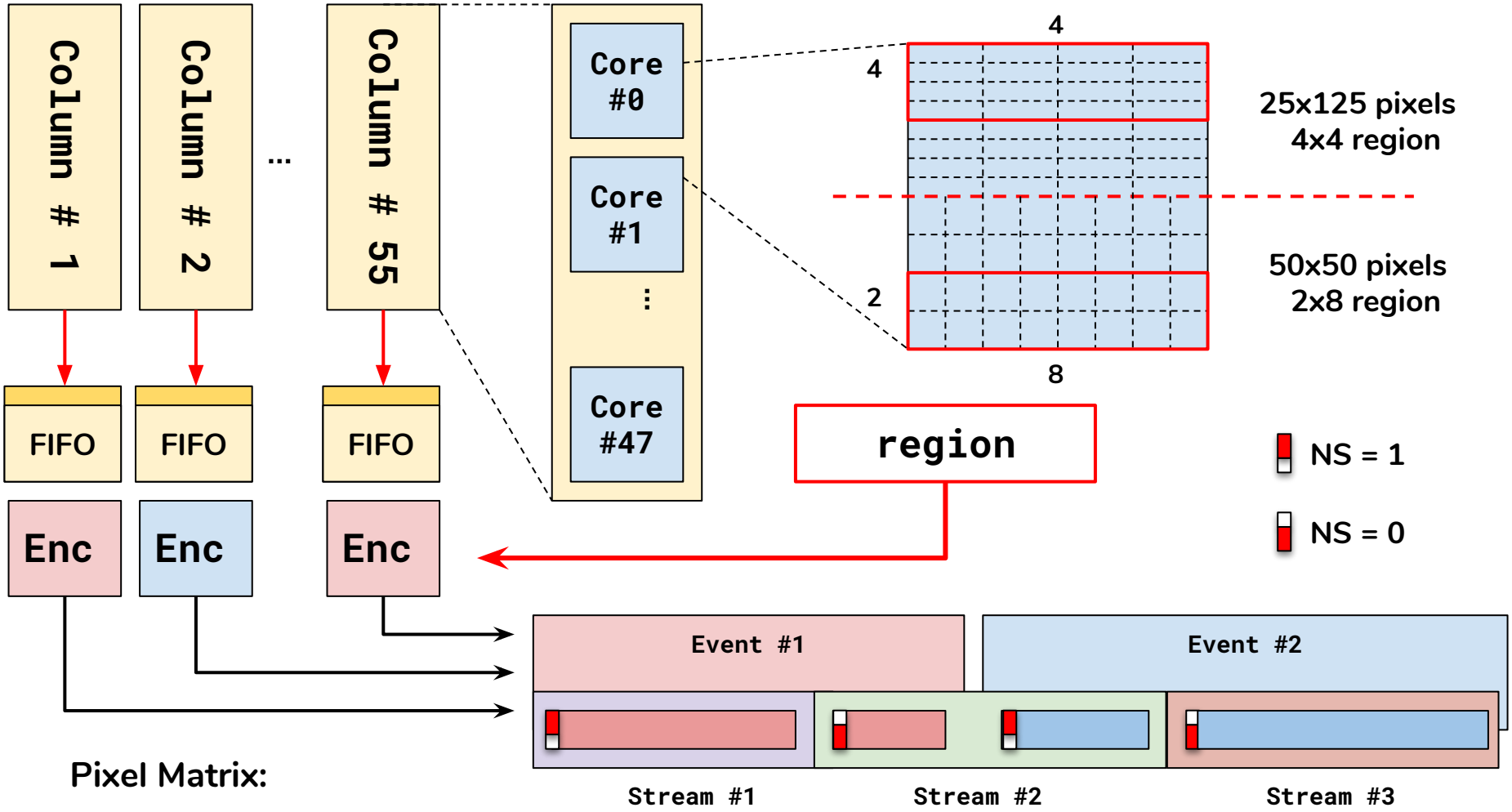
New Event(s) inside a Stream:

- In order to identify a new event we need to add the new BCId.
- As max ccol is **'110111'** we use **'111'** as a separator before adding the new BCId.

Ending a Stream each time an Event ends:

- There will be an option to enable this feature (very inefficient for Events with few Hits only).

Data inside the Pixel Chip



Pixel Matrix:

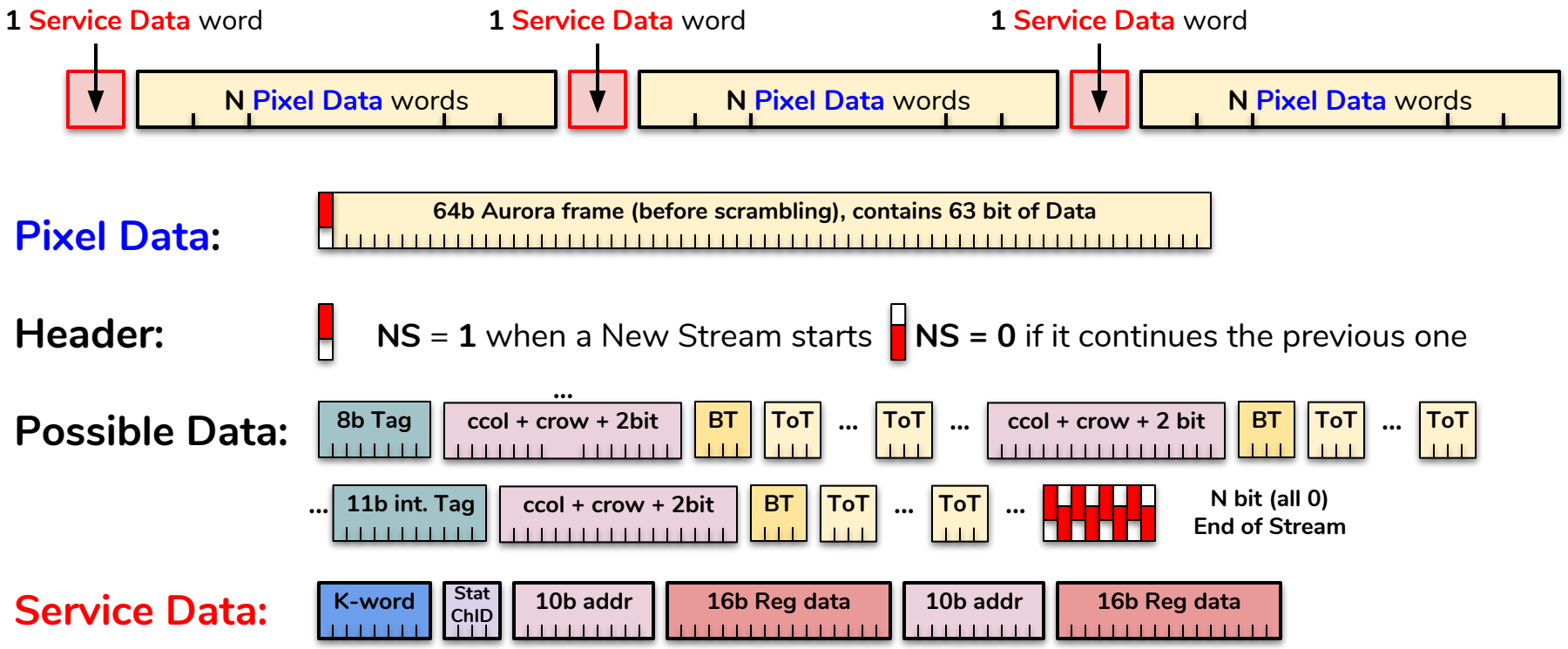
- All data are read out from Columns in parallel and are stored in EoC Fifo's.
- Binary Tree Encoding is performed on each region independently for each Column
- A full event is built preserving trigger order (for the time being NO event mixing is foreseen)
- Streams are filled with events and split over 64b Aurora Frames.

Output Data Stream [1, 2, 3, 4 links]

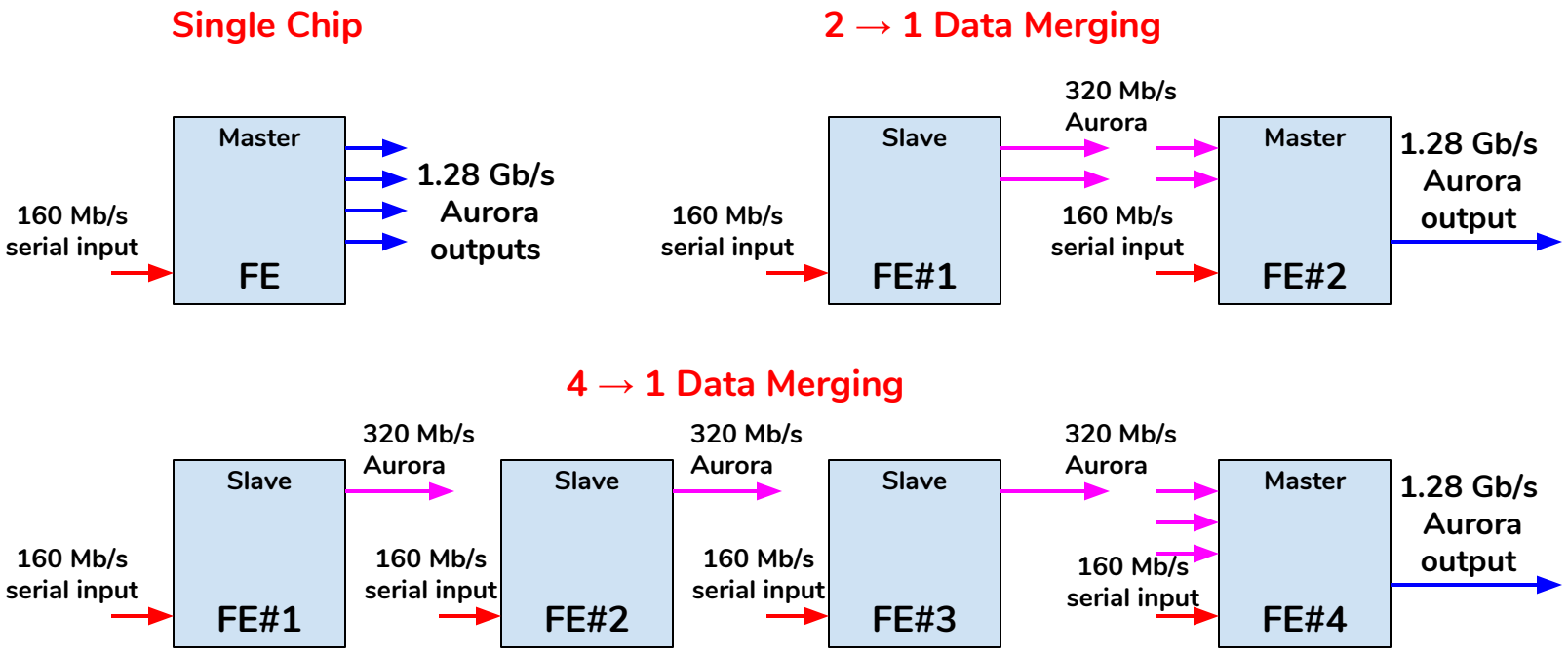
The Output Data Stream is sent on 1, 2, 3 or 4 different Aurora physical Lanes.
Each link is logically split in two separate channels:

Pixel Data and **Service Data** where the ratio between the two is user programmable.

1. Ratio **R** between Pixel Data and User Data is user programmable.
2. As an example a ratio R = (1 over 50) allows for ~100Mb/s of User Data

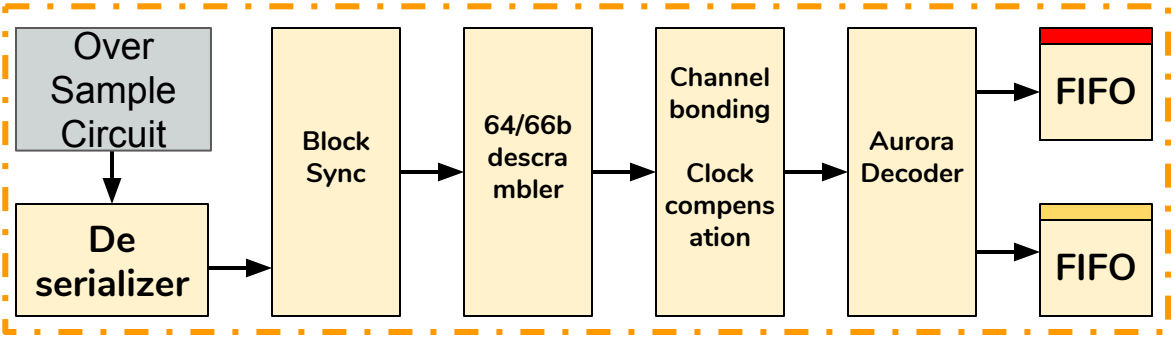


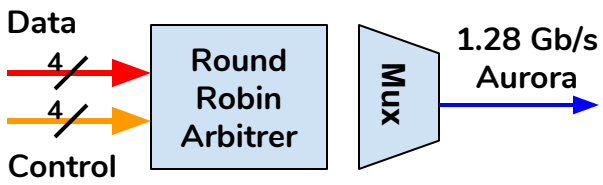
On chip Data Merging capability



- For outer layers though a single 1.28 Gb/s lane would be under utilized as there is not enough data to completely fill the available bandwidth.
- Depending on chip location in the detector a **2→ 1** or a **4→ 1** merging capability is needed.
- All links between chips will be **320 Mb/s** (possibly also **640 Mb/s**) Aurora 64/666b **single strict aligned lanes**.
- Possible configurations:
  - **A.** Single chip with up to four 1.28 Gb/s serial Aurora output lanes (5.12 Gb/s total output).
  - **B.** Two Slave chips and one master with three serial Aurora 320 (640) Mb/s inputs
  - **C.** Three Slave chips and one master with three serial Aurora 320 (640) Mb/s inputs

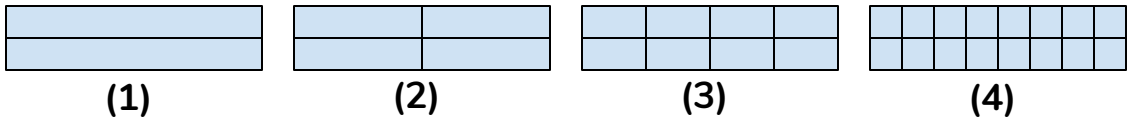
Receiver instance inside Master chip



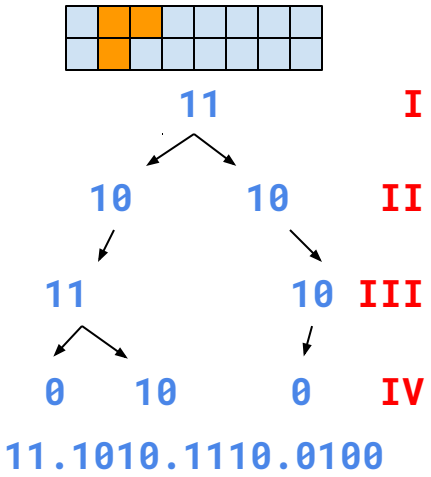


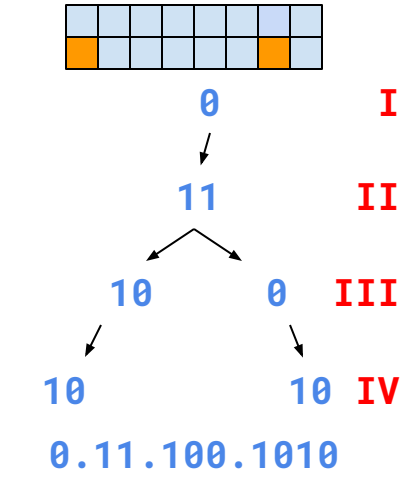
- Each incoming link needs a dedicated receiver instance (four in total) in order to correctly align, unscramble and decode the incoming Aurora stream(s)
- Only needed instances are active in the chip [user configurable option after startup]
- 1.28 GHz clock from CDR used to sample data from incoming link. Circuit (x3) inside CDR.
- Each Aurora stream consists of two distinct channels:
  - **Pixel Data:** (**01** header): Data coming from pixels
  - **Service Data:** (**10** header): Idle, Channel Bonding, User generated Data
- Data coming from each channel is stored in a separate Fifo
- Receivers are “dumb”, i.e. they do not have to know contents of decoded data.
- Need to identify ChipId of incoming data stream:
  - We will add 2 bit configuration to each Aurora packet [3% bw loss]
  - These bits will be put immediately after the StartOfStream bit
- Master chip will generate a complete **1.28 Gb/s link** using data stored in the Fifo’s Three Fifo’s coming from receiver and one for on chip generated data will be used.

Binary Tree Data Formatting



- Developed by Cornell group for CMS, it will be the data formatting used in RD53 for both experiments.
- **How it works**:
  1. TheCore Column (ccol) is a 6 bit wide mask that tell which column has data. No compression is done.
  2. CoreRow address (crow) is encoded with some compression. (compression details still to be defined).
  3. Once a 2x8 or 4x4 pixel region has been addressed a binary search is performed.
  4. The encoding is performed in 4 subsequent steps
  5. ToT information is added after encoded data. This allows for easy ToT data removal [will be an option].





The encoding has variable length, ranging from 4 to 30 bits.

Possible options inside the chip

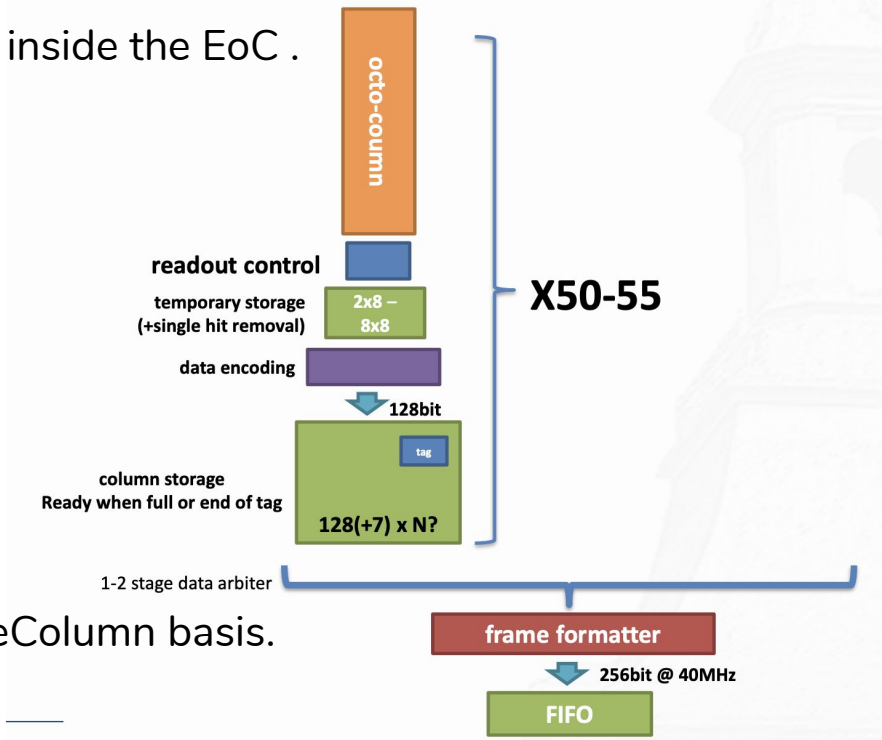
Isolated hit removal: - Will be performed, if enabled, inside the EoC .

Binary readout:

- In case of bandwidth problems we will be able To remove ToT values from Data Stream saving ~30% of bandwidth.

Data truncation:

- In case of very long events, or if Fifo's are getting full we will truncate events.
- The maximum number of events will be programmable and truncation will occur on CoreColumn basis.



Not encoded data output:

- There is a request to be able to readout data without passing through the encoder. [Debug]

Not ordered event fragments inside a Data Stream:

- Not foreseen for the moment. We will perform detailed simulation.

Self contained events inside a Data Stream:

- End a Stream each time and event ends. Can be easily implemented.

Data without Binary Tree Encoding: - Easy to implement. [Debug]

BCId and LV1Id inside streams: - An option will allow to add this information. [Debug]