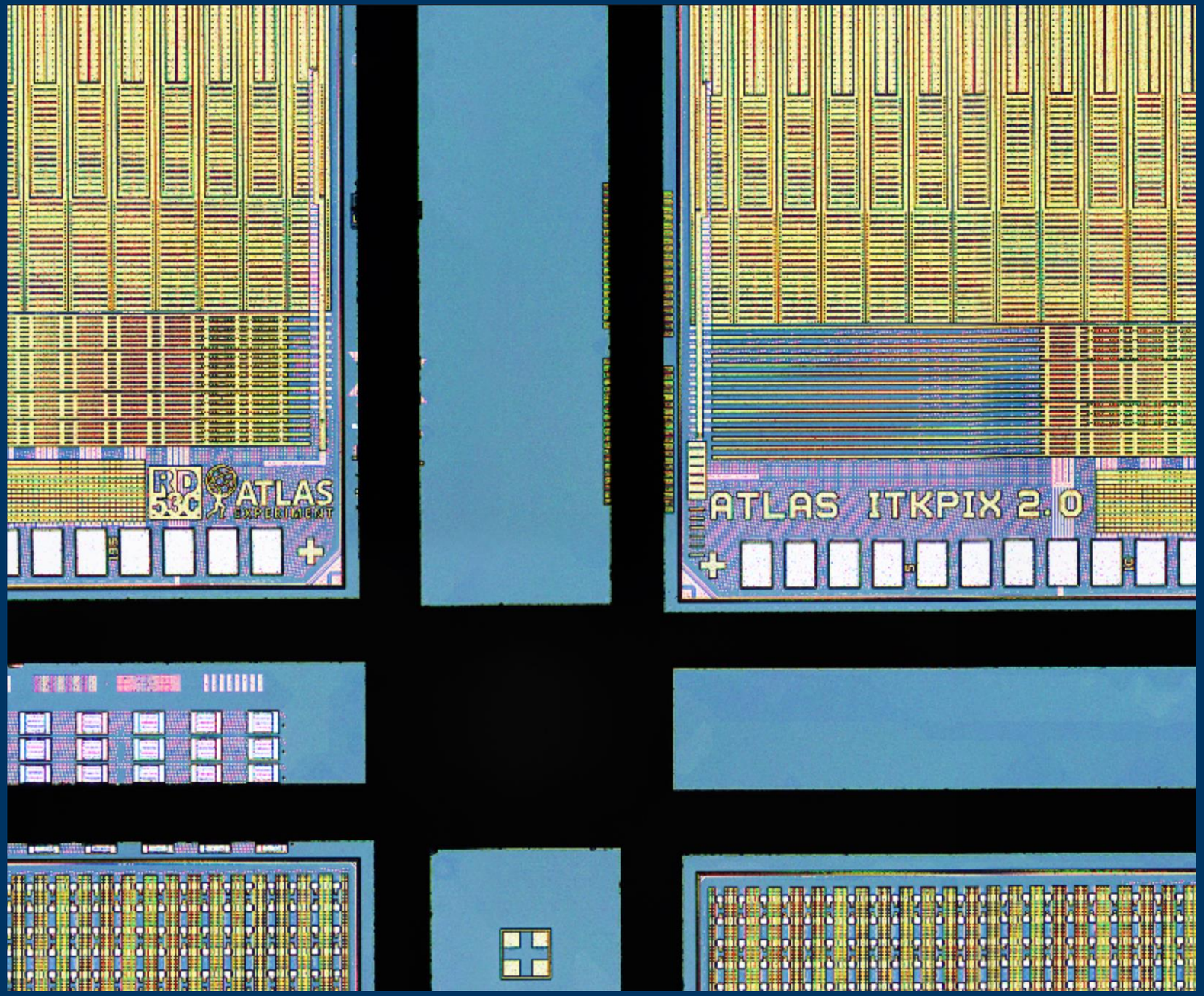
YARR Firmware Tutorial



Diced ITkPixV2 Readout Chips

Luc Le Pottier, Timon Heim

Lawrence Berkeley National Lab
January 14th, 2025

YARR SPEC Firmware Overview

- YARR uses custom PCIe firmware, on off-the-shelf FPGAs
- Recommended, best understood FPGA is **TEF-1001 / TEF-1002**
- YARR website contains documentation regarding setup, flashing firmware, etc.
- **What is SPEC?**
  - **S**imple (cheap)
  - **PCIE** (how we communicate with PC)
  - FMC (how we communicate with FEs)
  - **C**arrier

Trenz TEF-1001/2

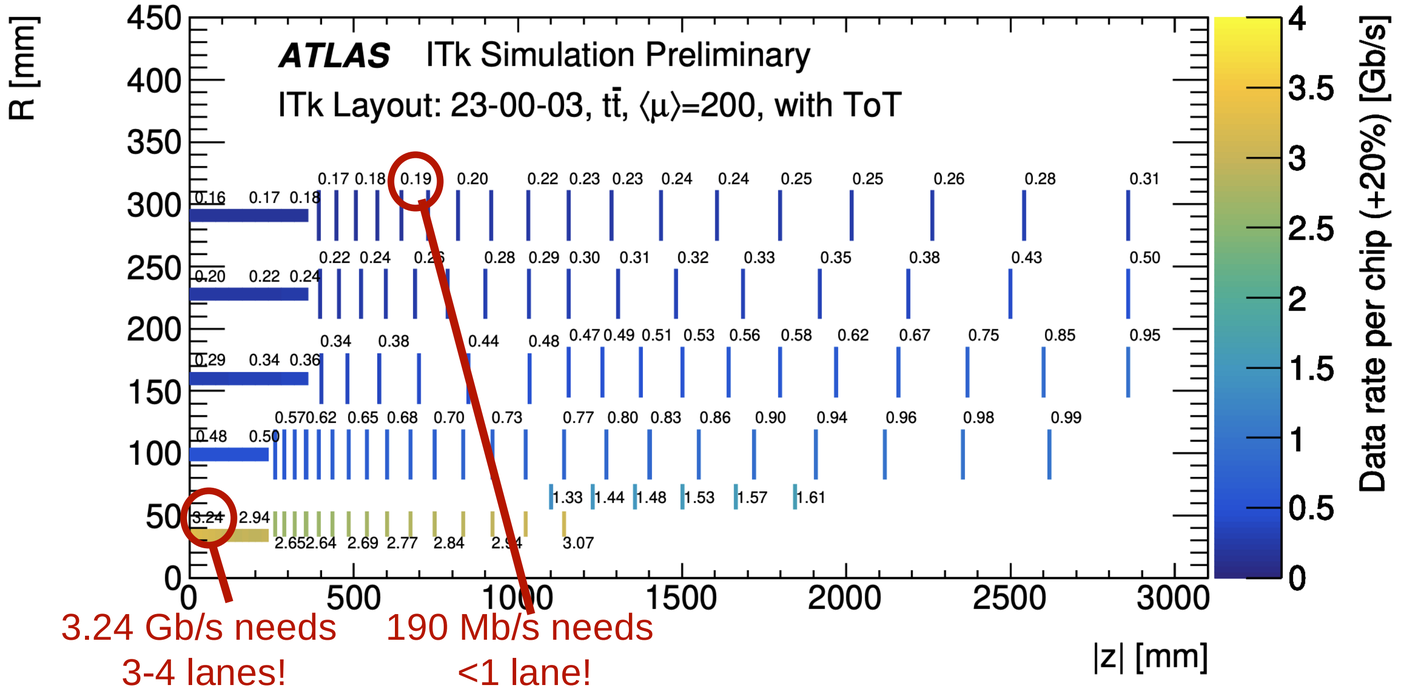
ReflexCES XpressK7

Xilinx KC705

Numato Nereid

Lanes and Data

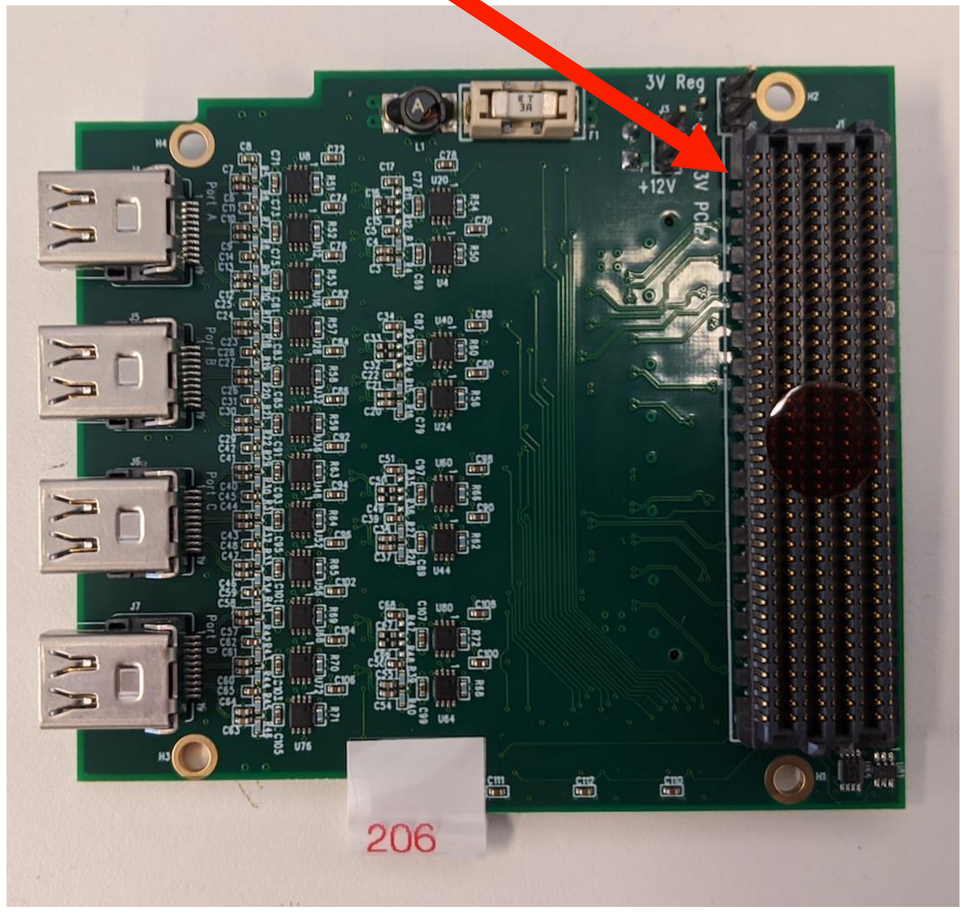
- Rd53 designed with large range of operating conditions in mind
- In particular: **20x difference** in bandwidth requirement between innermost / outermost layer chips!
- But, increased readout bandwidth = more power.. how to minimize?
  - 4 readout channels per chip, called **lanes**
  - Maximum of **1.28 Gbps** on each lane
  - Lanes will be disabled if not needed
- **Examples:**
  - Outer layer quad with 160 Mb/s/chip needs 4x160 = 640 Mbps, fits on a single lane (can be combined with **data merging**)
  - Inner layer quad with 3.24 Mb/s/chip needs **3-4 lanes per chip** (4 to hit S.F.)

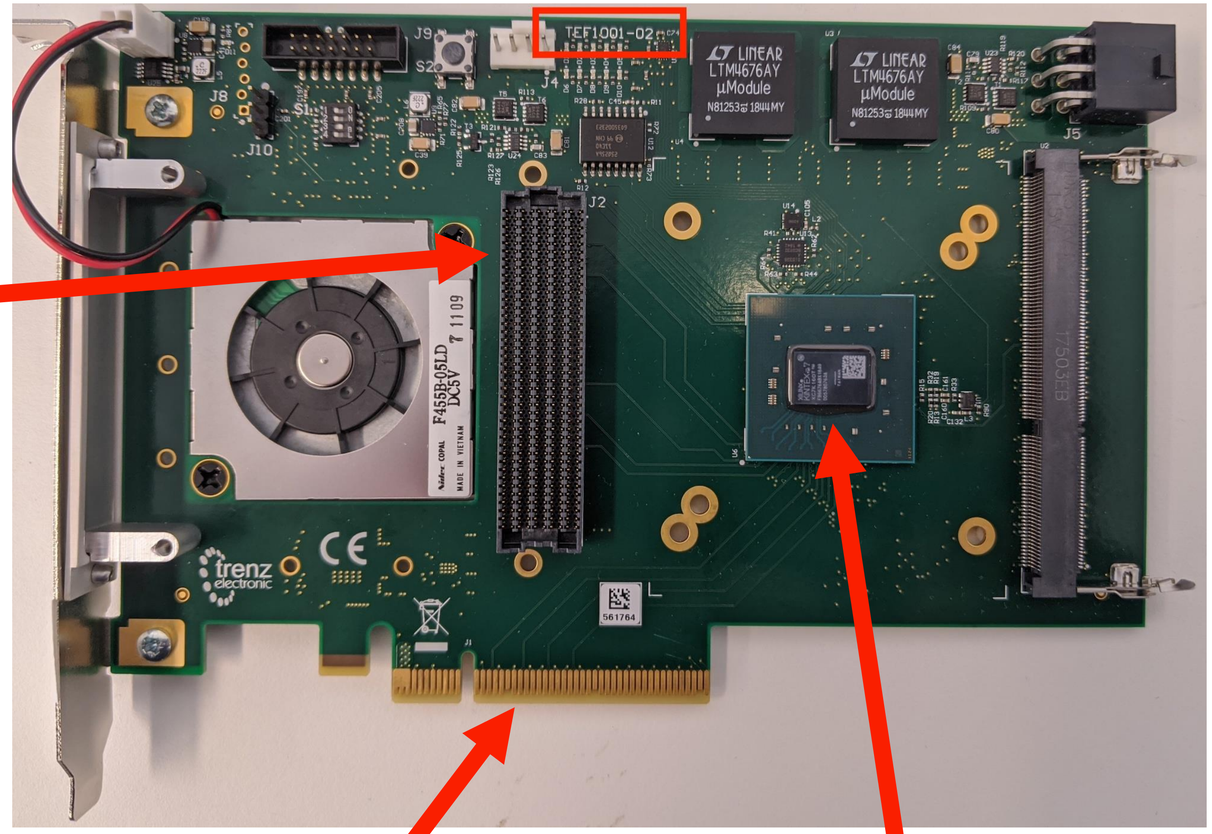


Inner pixel system simulated data rates per chip (ATL-ITK-PUB-2022-001)

Boards

- **Ohio card** provides 4 display ports for connection to FEs, each with capacity for **4 lanes per port,** for 4x4 = 16 lanes
- **FMC connector** bridges the Ohio cards and FPGA





PCIe connector

Kintex 7 FPGA

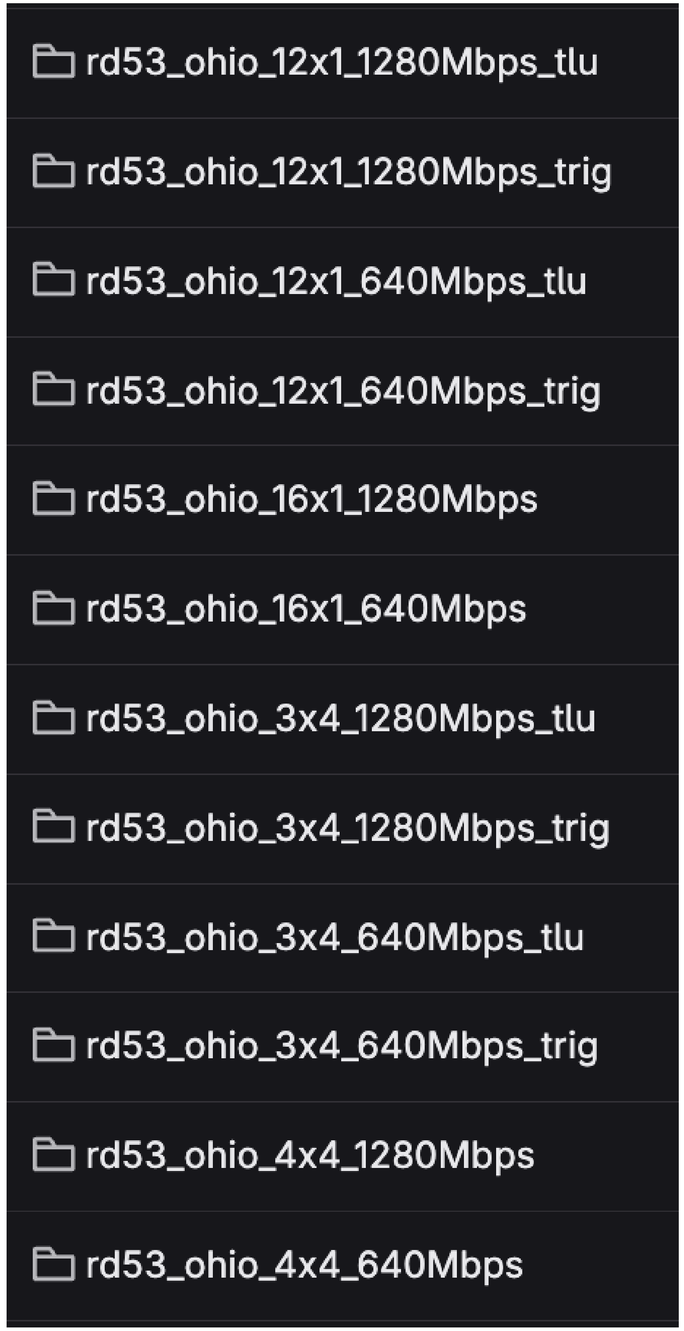
- Each port contains **four data lanes** and **one command** lane
- For **external trigger** or **TLU** firmware, port D may be modified to accept other signals

Boards + versions

- Firmware is according to the lane configuration we are expecting
  - **4 x 4** means 4 ports, each with 4 lanes enabled per chip
    - **Required** to run at **> 1 Gbps** data rate for a single chip
  - 16 x 1 means 4 ports, each with 1 lane enabled per chip
    - All four lanes on a port may have a FE attached, as is the case with standard quad modules
    - **Required** for running quad modules without data merging

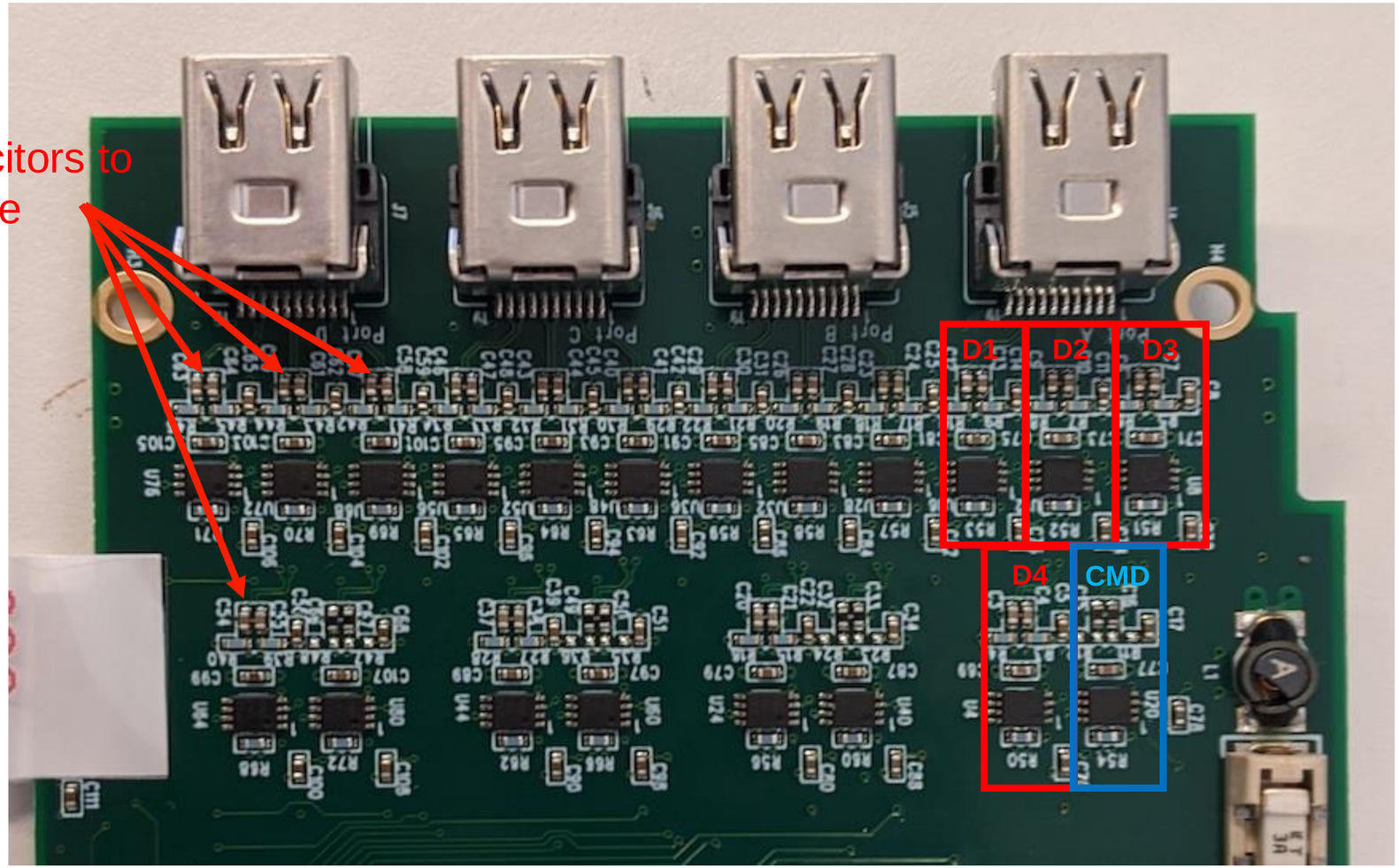
- **640/1280** Mbps indicates the bandwidth for each lane
  - Generally **1280** Mbps is more susceptible to typical high speed transmission problems from e.g. long cables
  - But, 640 does not maximize readout rate

- Suffix indicates whether pins for port D are mapped to TLU/trig



Boards + Triggers

- Capacitors C54, C55, C57, C58, C60, C61, C63, C64 should be replaced with 0 Ohm resistors
  - Modification changes AC coupling to DC coupling
- This allows port D to receive **external triggers**, or TLU signals (important for test beam)
- All mappings between pins and hardware can be found in the **.xdc files** for a given firmware version



Boards + Debugging

Ohio cards can have problems..

- **Incorrect modifications** (or lack thereof)
  - All data + CMD lines should be AC coupled with 50 ohm termination
  - All trigger inputs should be DC coupled
  - Causes readout errors, data processor complains
- **Loose mini display ports**
  - Soldering is not always consistent
  - Causes readout errors, data processor complains
- **Bad FMC connection**
  - Big connector with lots of pins -> some are too loose, unplug and plug back in
  - Causes readout errors, data processor complains

Boards + Debugging

Ohio cards can have problems..

- **Incorrect modifications** (or lack thereof)
  - All data + CMD lines should be AC coupled with 50 ohm termination
  - All trigger inputs should be DC coupled
  - Causes readout errors, data processor complains
- **Loose mini display ports**
  - Soldering is not always consistent
  - Causes readout errors, data processor complains
- **Bad FMC connection**
  - Big connector with lots of pins -> some are too loose, unplug and plug back in
  - Causes readout errors, data processor complains

Hardware issues may manifest visibly in software!

Configuration and Reset

Firmware uses the **wishbone interconnect architecture**

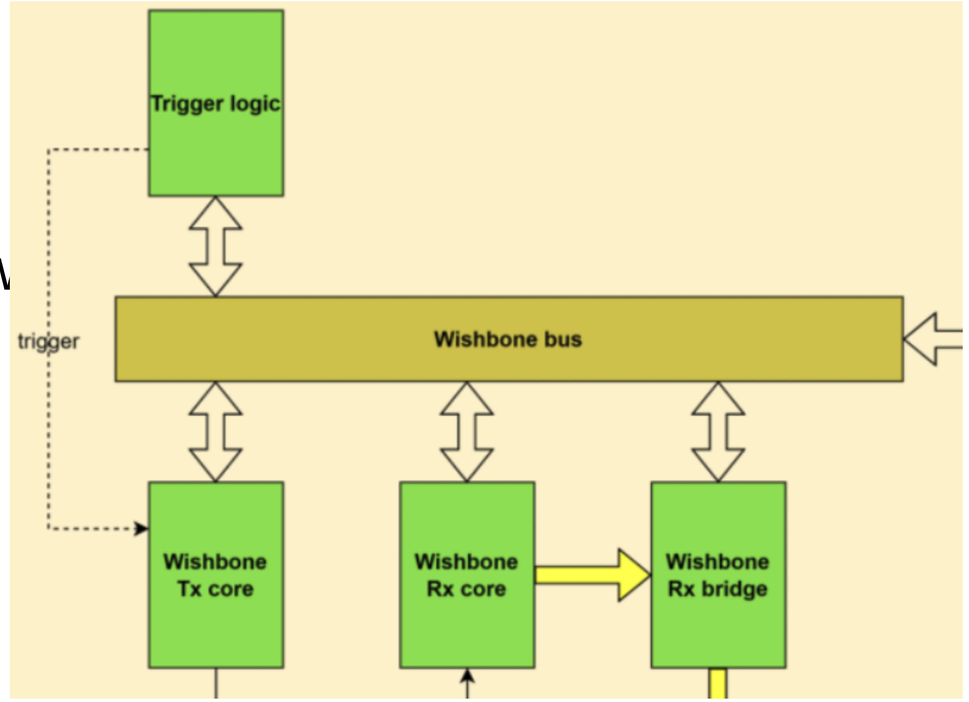
- Provides a way for different **IP cores** (chunks of firmw
- I.e. **PCIe** and **FIFO** implementations

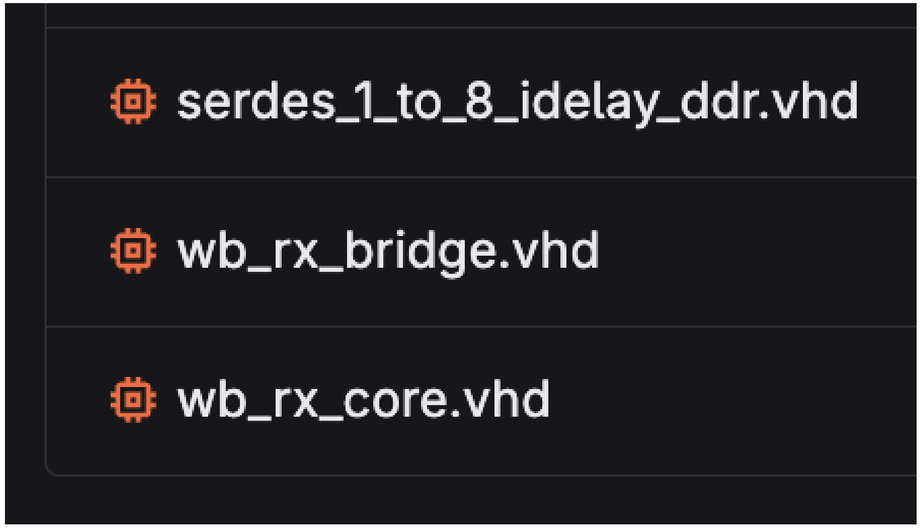
Wishbone buses provide

- **Firmware config** registers
- Local “soft” **resets**
- Block RAM transfers to PCIe

Configuration for FW typically goes through wishbone

- Rule of thumb 1: files with the suffix “**wb**” have a wishbone interface
  - This means they can easily be modified to take new configuration parameters
- Rule of thumb 2: firmware configuration comes from the **controller config file** in YARR: “-r”



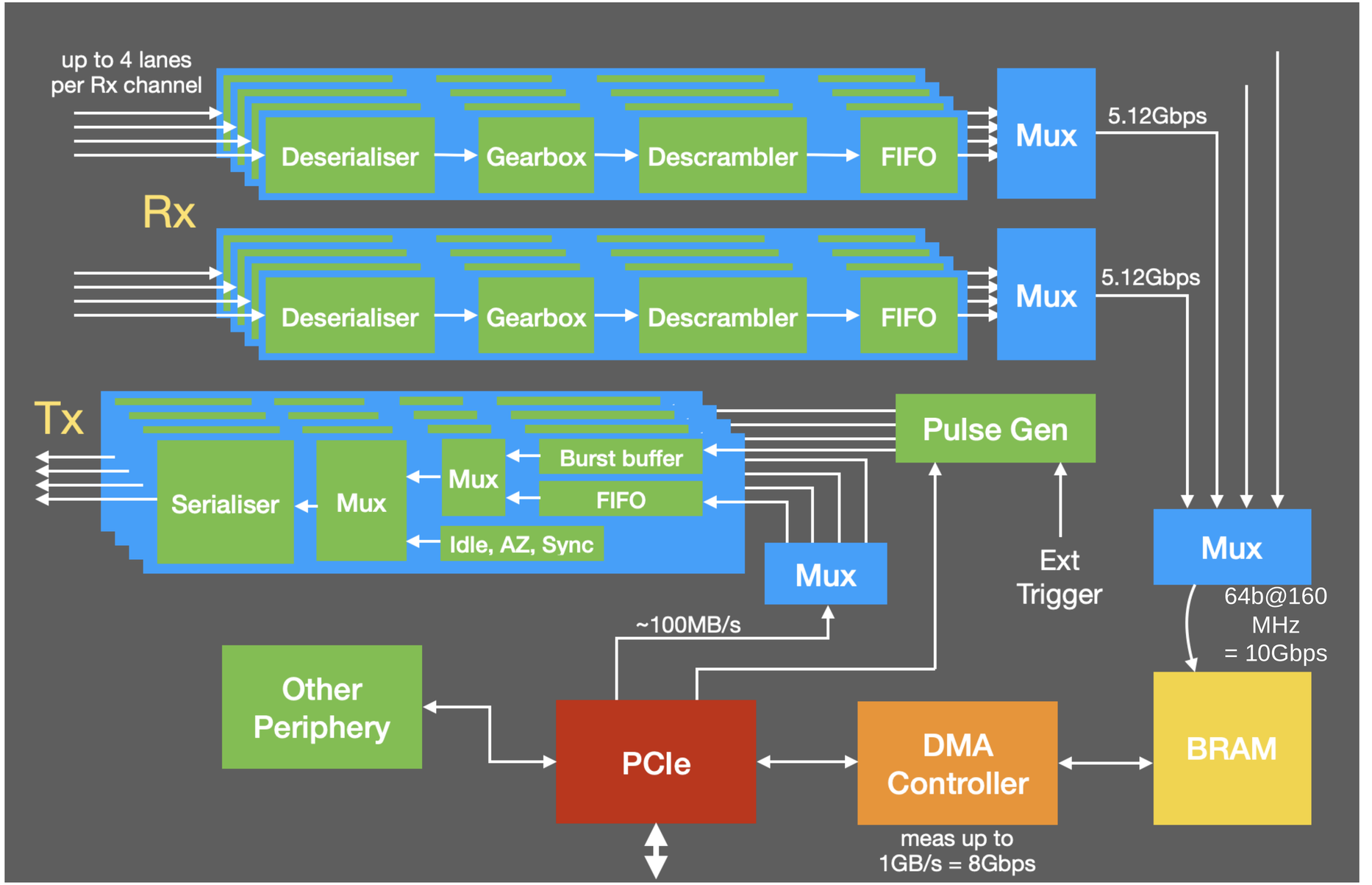


Configuration and Reset

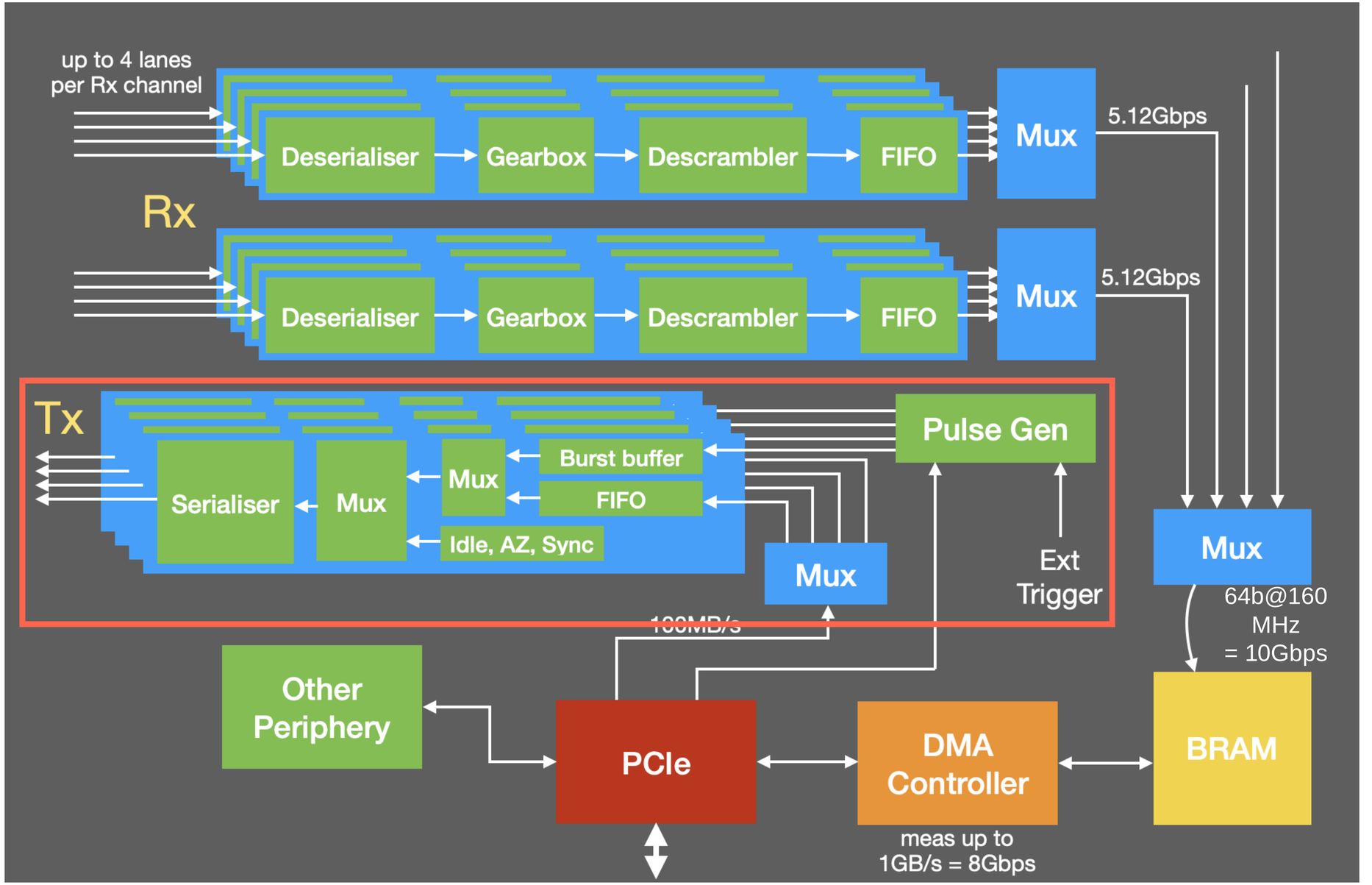
Useful wishbone related YARR scripts:

- `specSoftReset`: sends soft resets to wishbone modules, usually helping with run-time firmware issues
  - To reset all registers: `./bin/specSoftReset –r <controller_file> -o 15`
- `specComTest`
  - reads back firmware version / lane configuration, and tests basic transfers to FPGA
  - Useful to ID firmware version for debugging, or check that flashing has worked
  - Typical usage: `./bin/specComTest <spec_id>`
- `specReadWriteReg`
  - Allows for reading/writing of wishbone registers from the command line
  - **Expert level tool**, but documentation of registers for the future can be very useful
  - Typical usage: `./bin/specReadWriteReg –r <controller_file> -b 32768 –o 1`
    to check 8-bit RX link status for all four RX lanes
    (should return 0x3030303 for full function)

Data Walkthrough

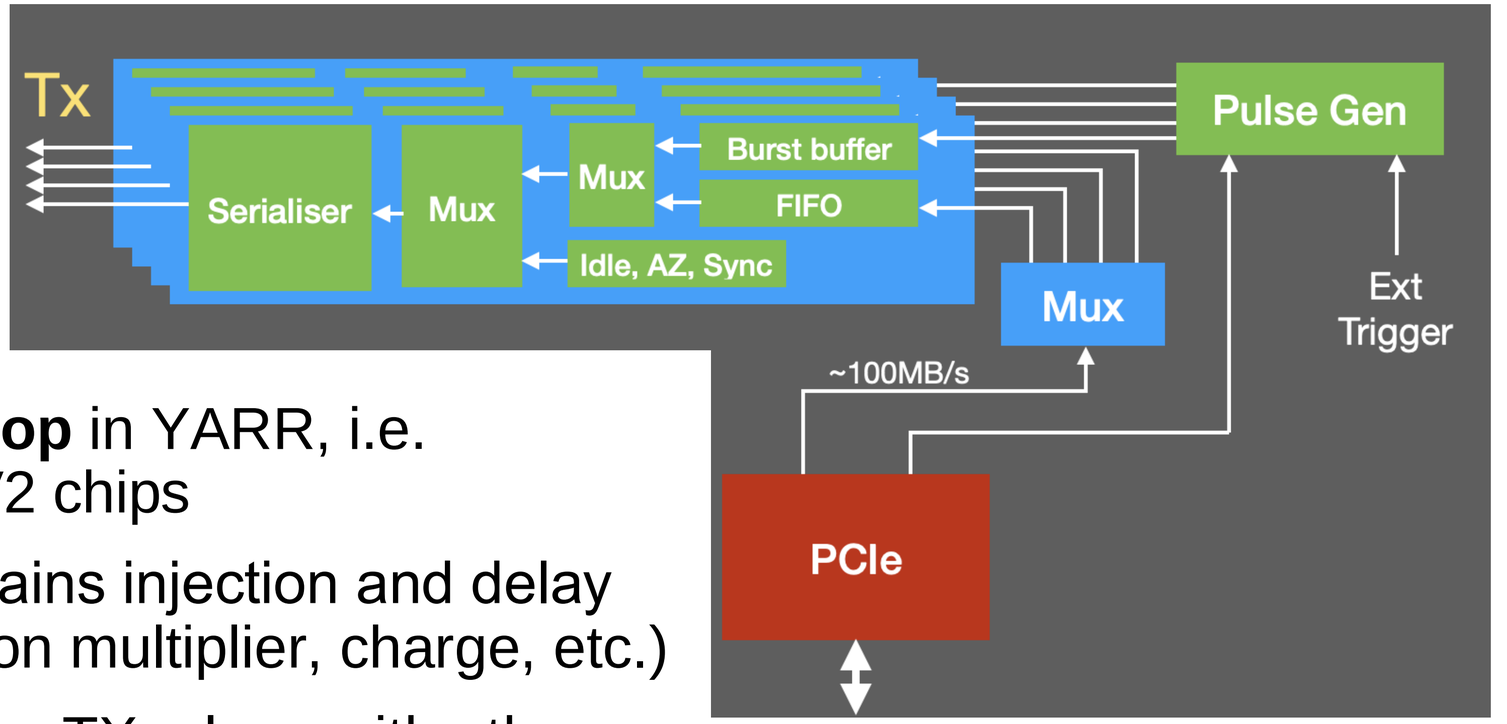


Data Walkthrough



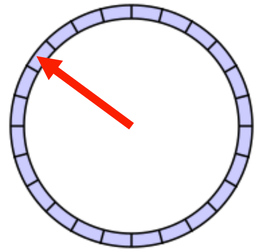
Triggering

Triggering



For typical scans…

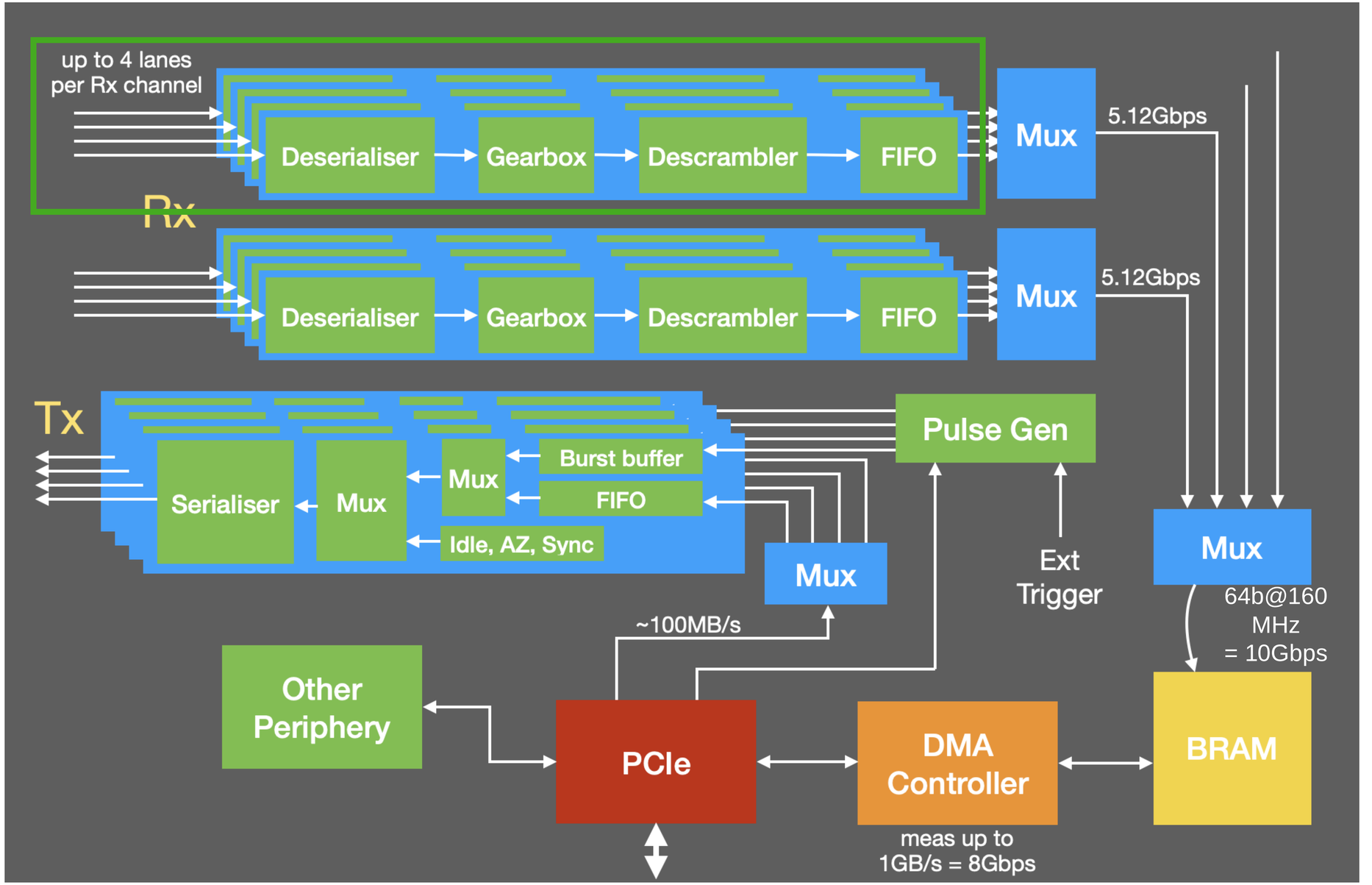
- Triggers are built in **<FE>TriggerLoop** in YARR, i.e. ItkpixV2TriggerLoop.cpp for ITkPixV2 chips
- 32 bit “trigger word” that is built contains injection and delay commands for the chip (dependent on multiplier, charge, etc.)
- This word is sent via **wishbone** to the TX, along with other information about target scan **frequency**, **duration**, number of **iterations**, etc.



In the FPGA…

- Pulse generator + trigger logic blocks begin cycling through the trigger word buffer repeatedly, at the specified frequency and for the specified duration/iteration number
- Can also use **external triggers** by enabling them properly from software
  - Trigger tags: new addition, built in firmware as we go
- Serializer sends commands to all enabled FEs simultaneously

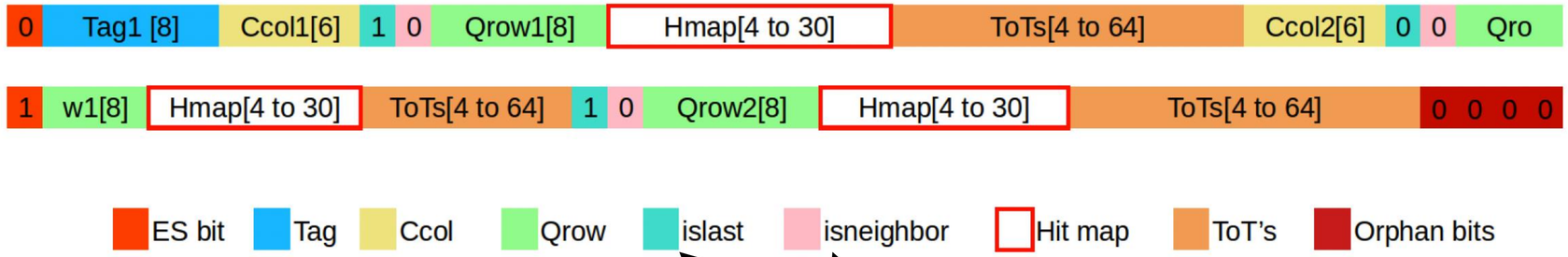
Data Walkthrough



Encoding + De-serializer

Encoding

How does the chip package hit data?



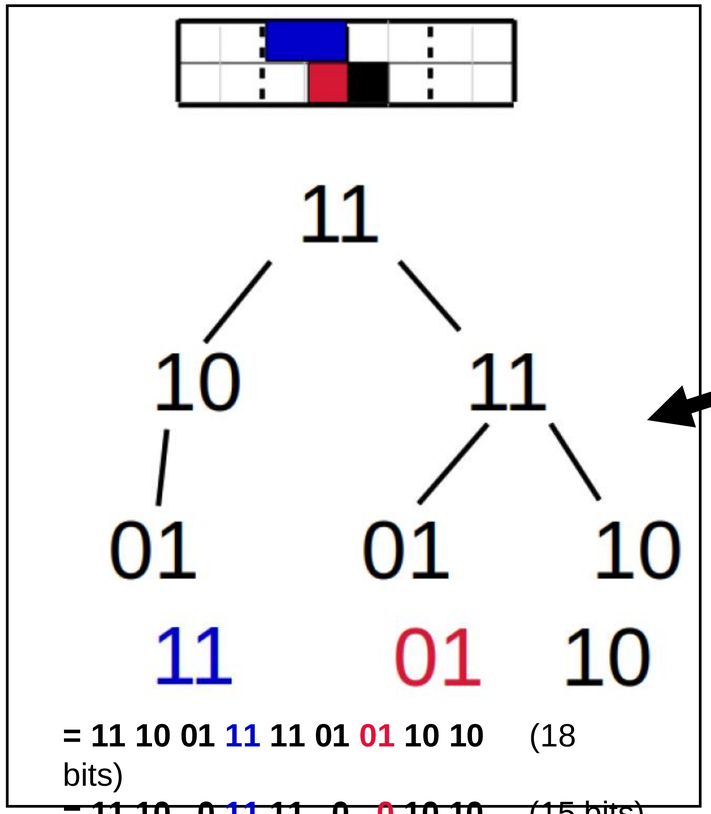
Encoded output for three hits, two of them being in the same core column, spanning two 64 bit data blocks

MIPs form clusters

We **drop addresses** by identifying neighboring pixels

Binary tree encoding of hits in a quarter core

01 -> 0 substitution everywhere

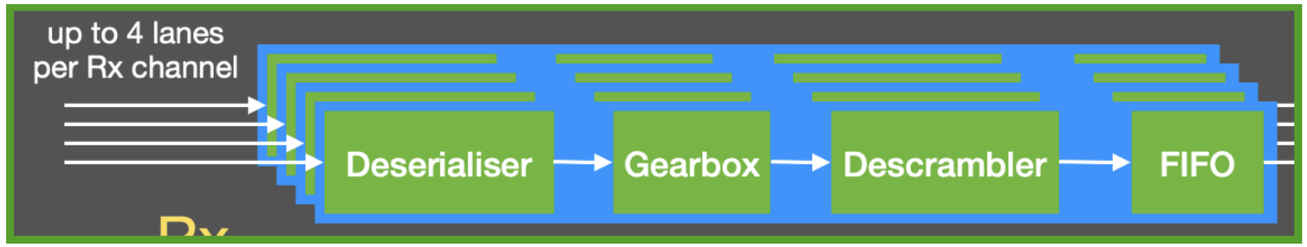


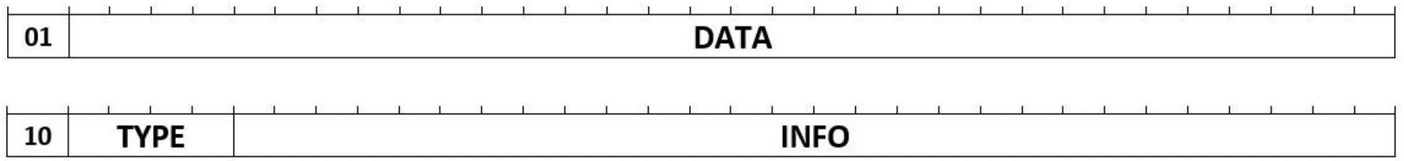
RD53 Encoding Scheme for a four-hit event within a single quarter core

Highly compressed, but **extremely complicated**

- An event with 50 hits could range in encoded size by 2-3x
- Packaged into **64 bit data blocks**, in as many as are needed for a full event
- Not **byte aligned**: no fixed event size

De-serializer

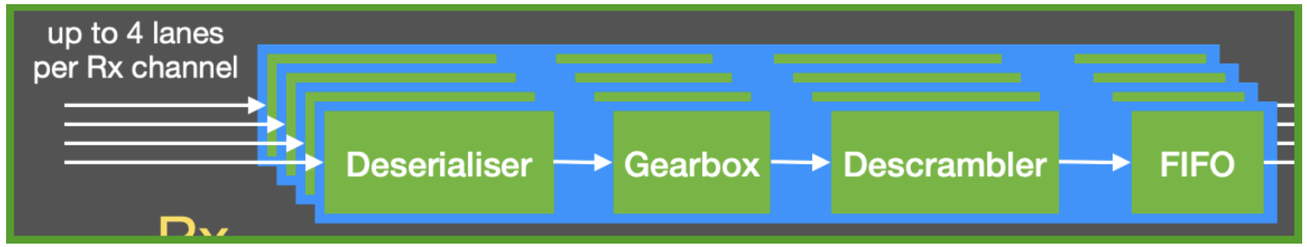




- Rd53 sends data using the [Aurora 64b/66b encoding]
  - Two identifying bits for either **data** (01) or **info** (10), where info is service information (auto/read registers)
  - Rest of the data package is **scrambled**
  - Any other value means there is an error, and the entire payload is lost
  - Pros of this encoding: bit-balanced payloads, **clock recovery** is possible, **stream alignment** is possible, fast enough (1.28Gbps)

- **Problem:** if we have a bit shift in the data stream, we need to resynchronize
  - Involves looking at consecutive streams until we get **01** or **10** consistently
  - Takes a long time and loses **all data** in the 64bit packets
  - **One bit wrong** = **whole word wrong**

De-serializer



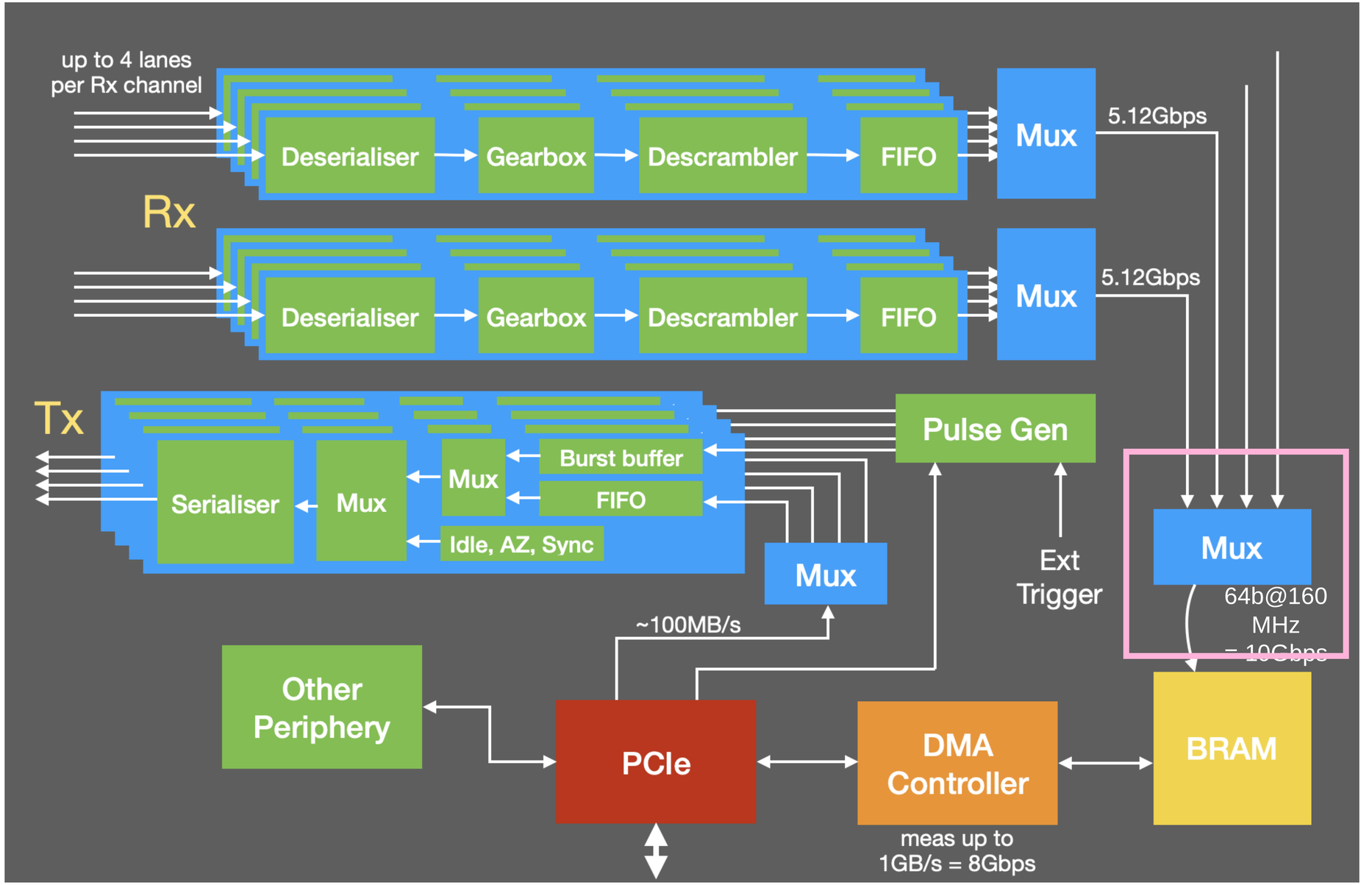
- From there, data is synchronized on a lane-by-lane basis, and **idle words** are removed from lanes without data

- It is then multiplexed per-lane such that each channel contains a single output stream, with a maximum bandwidth of 4.96 Gbps

- Lastly, all lanes are passed into their own FIFOs
  - This is because we typically expect temporary **high occupancy** during a trigger, followed by no trigger for a while, so buffering is useful before possible bottlenecks

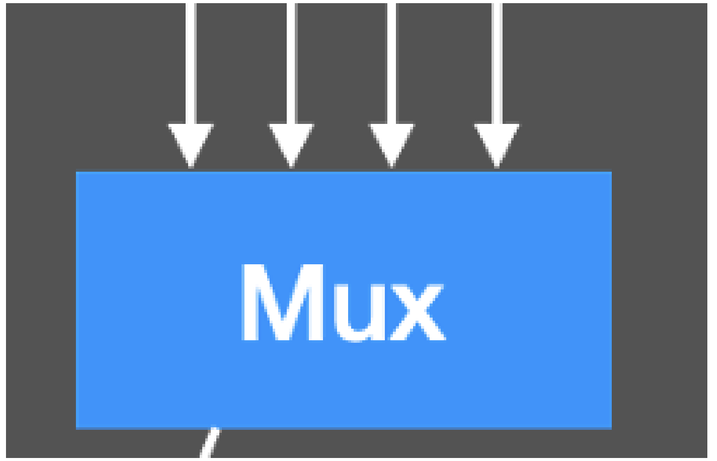
    (HL-LHC trigger rate of 1MHz vs. BC rate of 40MHz)

Data Walkthrough



Channel Multiplexer

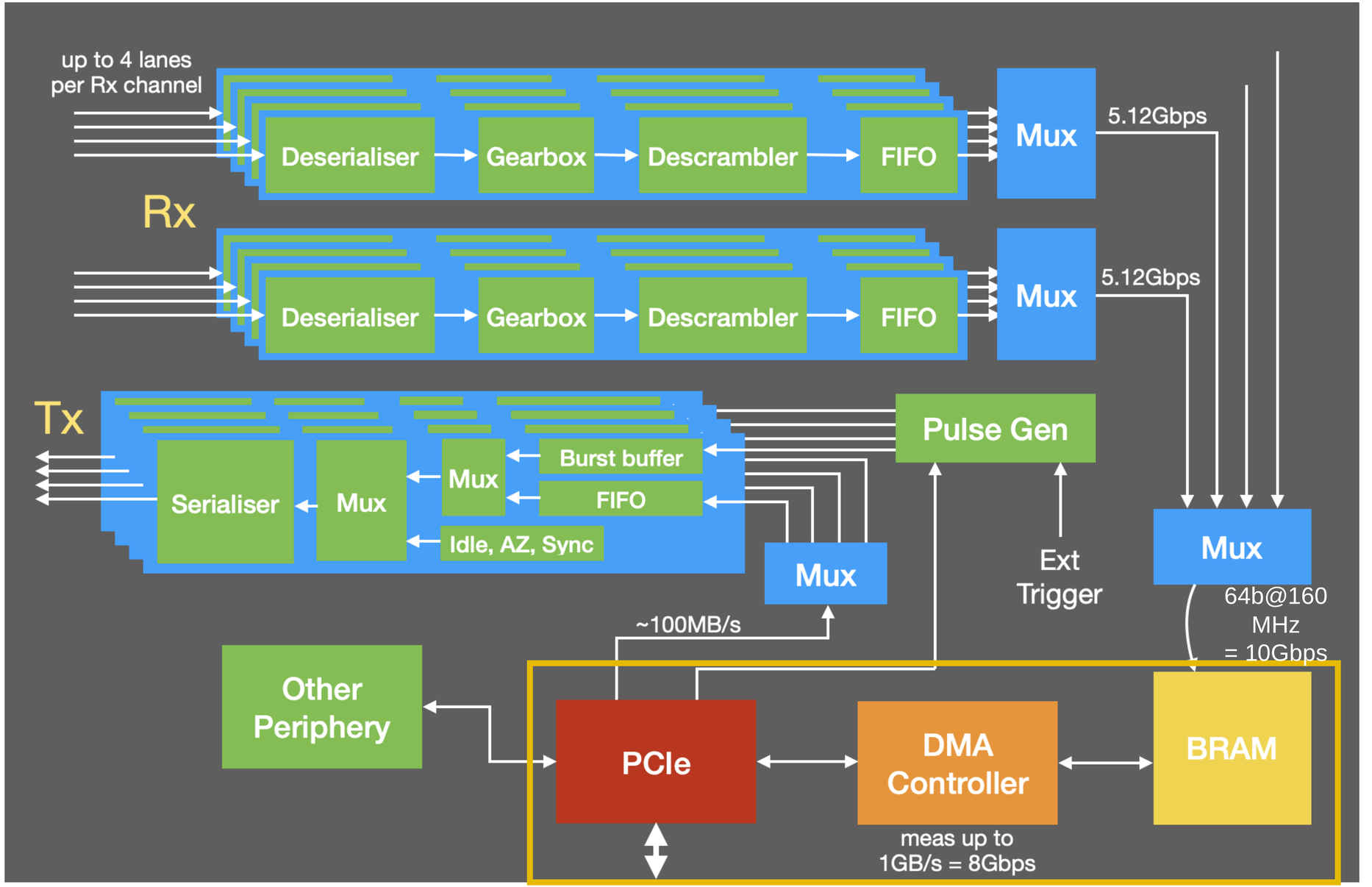
Channel Multiplexer



- Each of the four **channels** (corresponding to ports on the Ohio card) go to a **round-robin arbiter**
  - This block multiplexes the data into a single stream
  - Since different channels have different volumes of data, need a way to decide on which order to add them to the final output stream as they are coming in
  - It is also a system bottleneck: 160MHz clock limits the data rate to 160*64 = **10.24Gbps**, lower than the 19.86Gbps previously
  - (FW upgrade to make this clock 320 MHz, but it is complicated)

- **Possible issues:**
  - Empty words added during arbitration, for channels without data -> blows up data rate
  - Can occur during self-trigger scan if there are very **noisy pixels**
  - In particular: “noisy” from noisescan != “noisy” in selftrigger scan… hard to mask with a typical noise scan.

Data Walkthrough



Block RAM

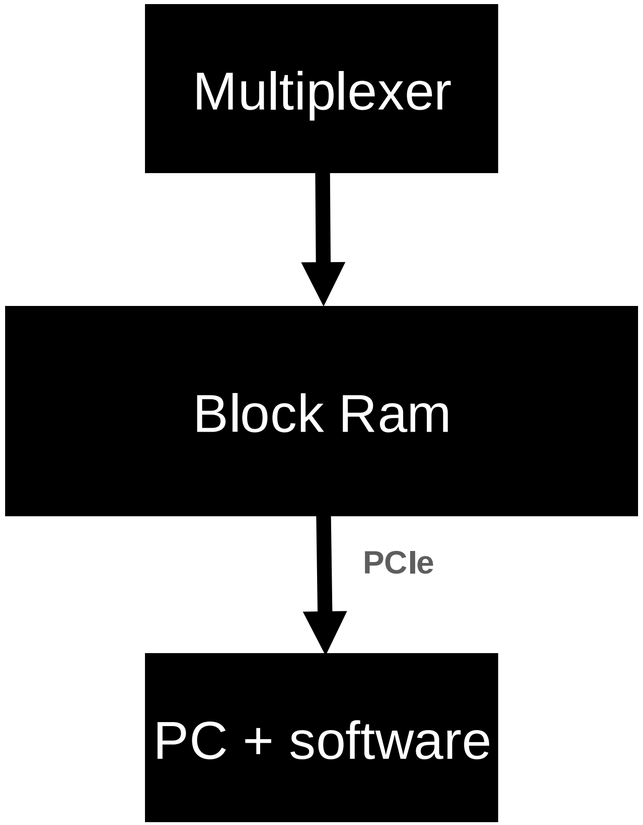
Block RAM

Multiplexed data for all **four channels**, each with up to **four lanes**, is passed to the **Block RAM** (BRAM)

- 64 rows by 16 columns of 18kb RAM blocks = 18 Mb of memory
- Batches data for fast block transfer to PC over PCIe

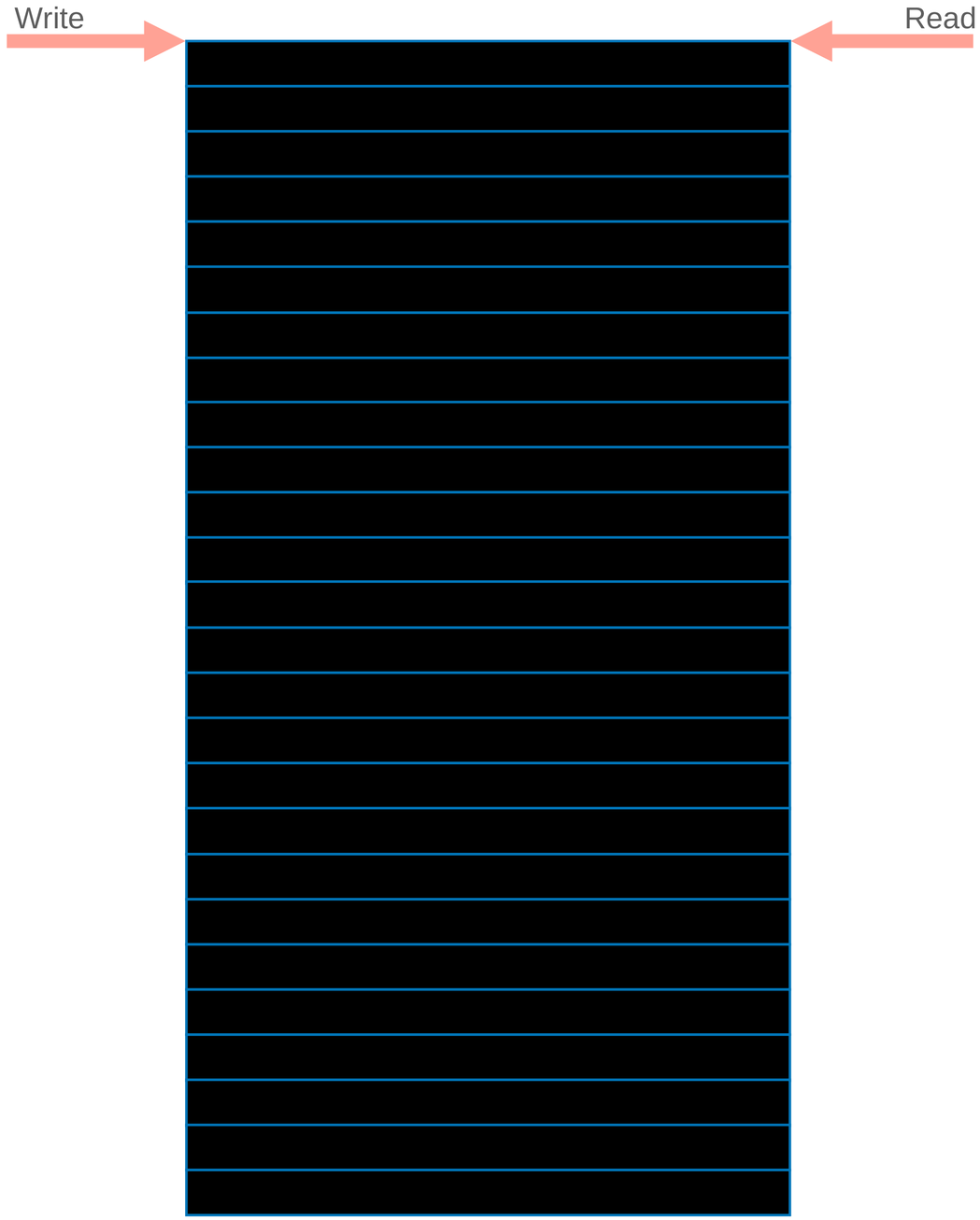
Possible problems:

- If **data in** > **data out** of the block ram, we will start to overwrite data that has not been read out
- This can happen when…
  - We have a very high data rate (**data in** is fast)
  - We have a very slow PC (**data out** is slow)
- We implemented a throttle mechanism for the BRAM, which helps during FPGA-triggered scans, but:
  - Cannot throttle **self-trigger** scans
  - Does not cover case where **PC reads** stop for a “long” time



Block RAM

What does this look like?

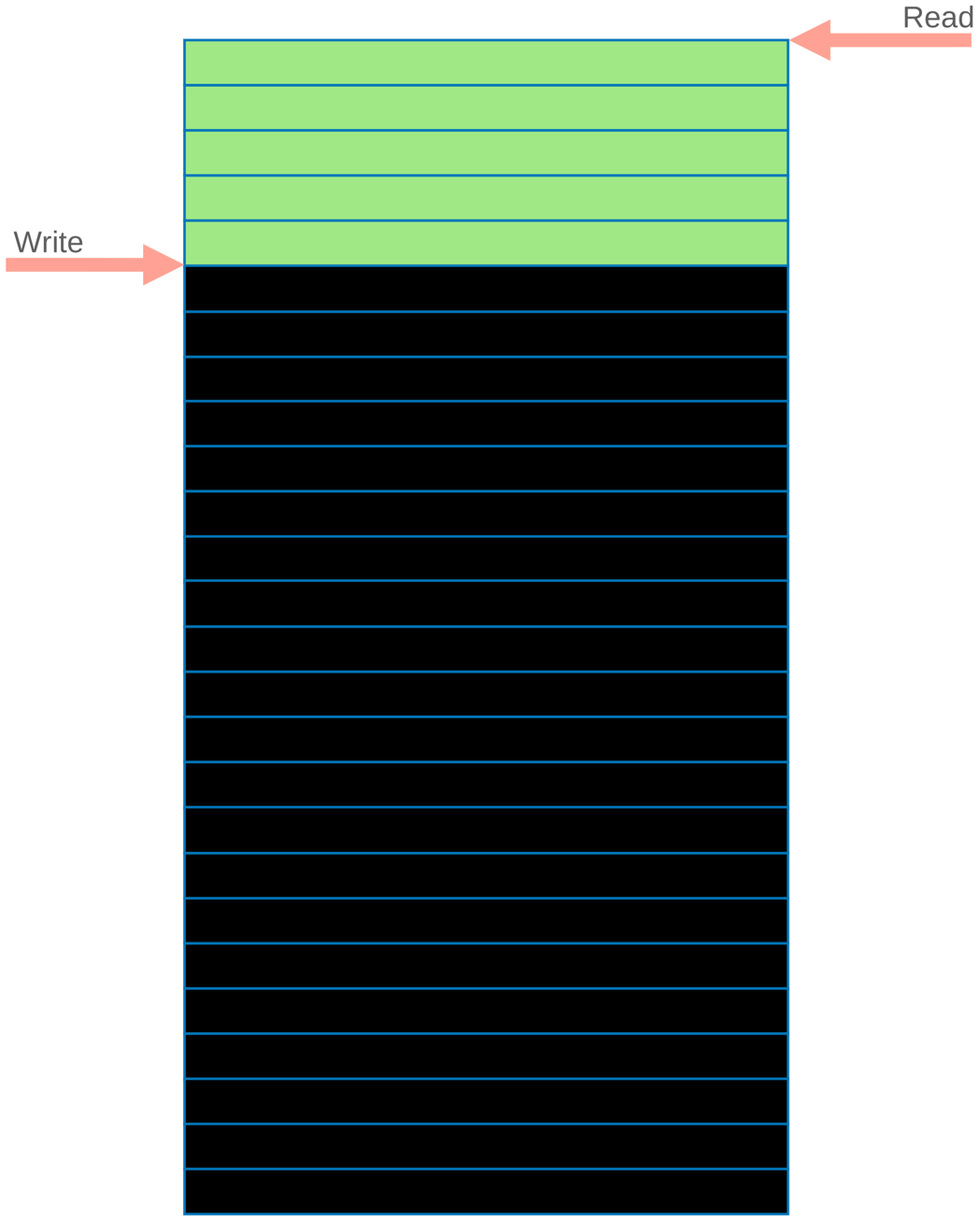


BRAM simplified schematic

Block RAM

What does this look like?

- Data for an event is written to the BRAM upon reception

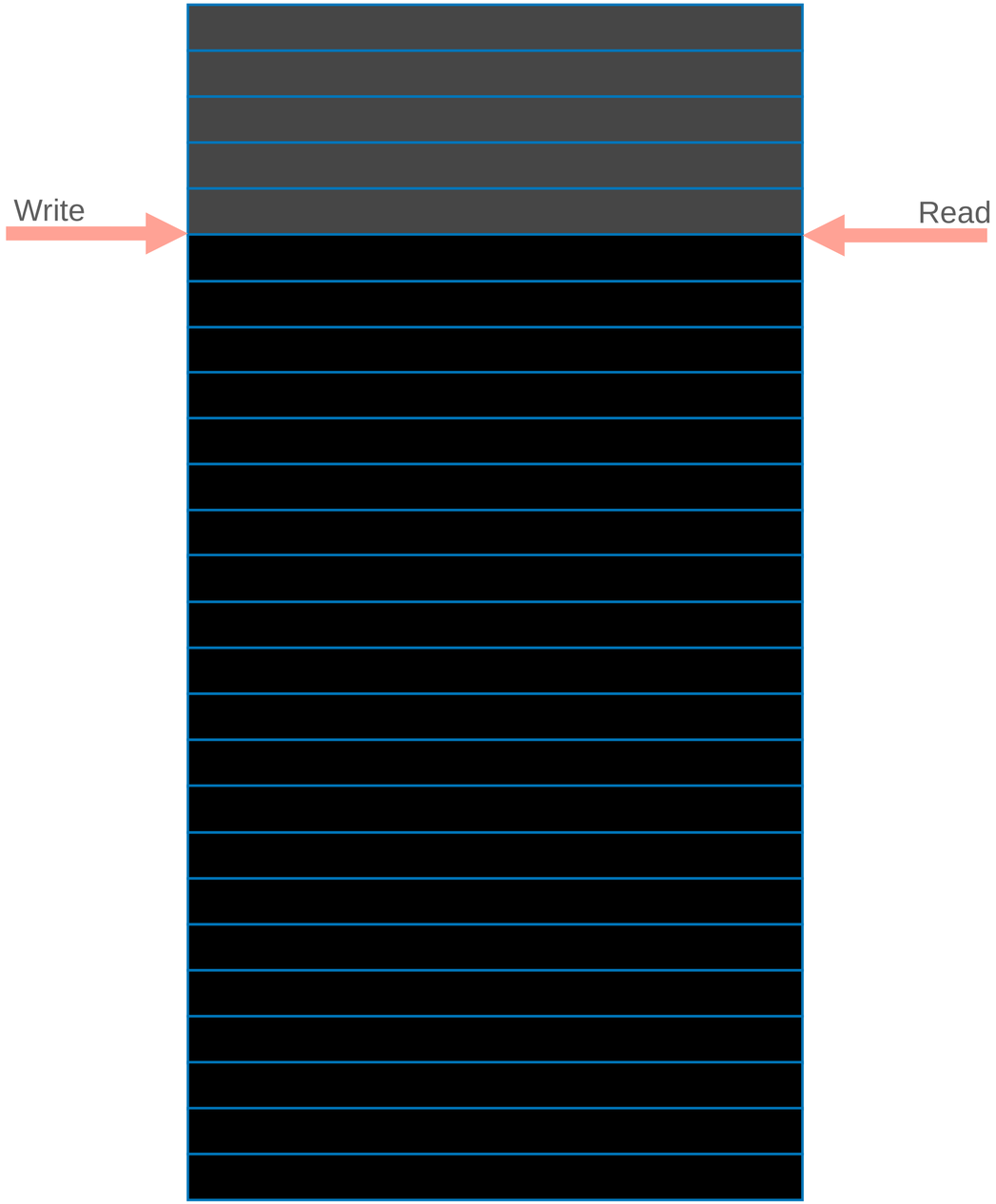


BRAM simplified schematic

Block RAM

What does this look like?

- Data for an event is written to the BRAM upon reception
- Data is read by PC as it is able to

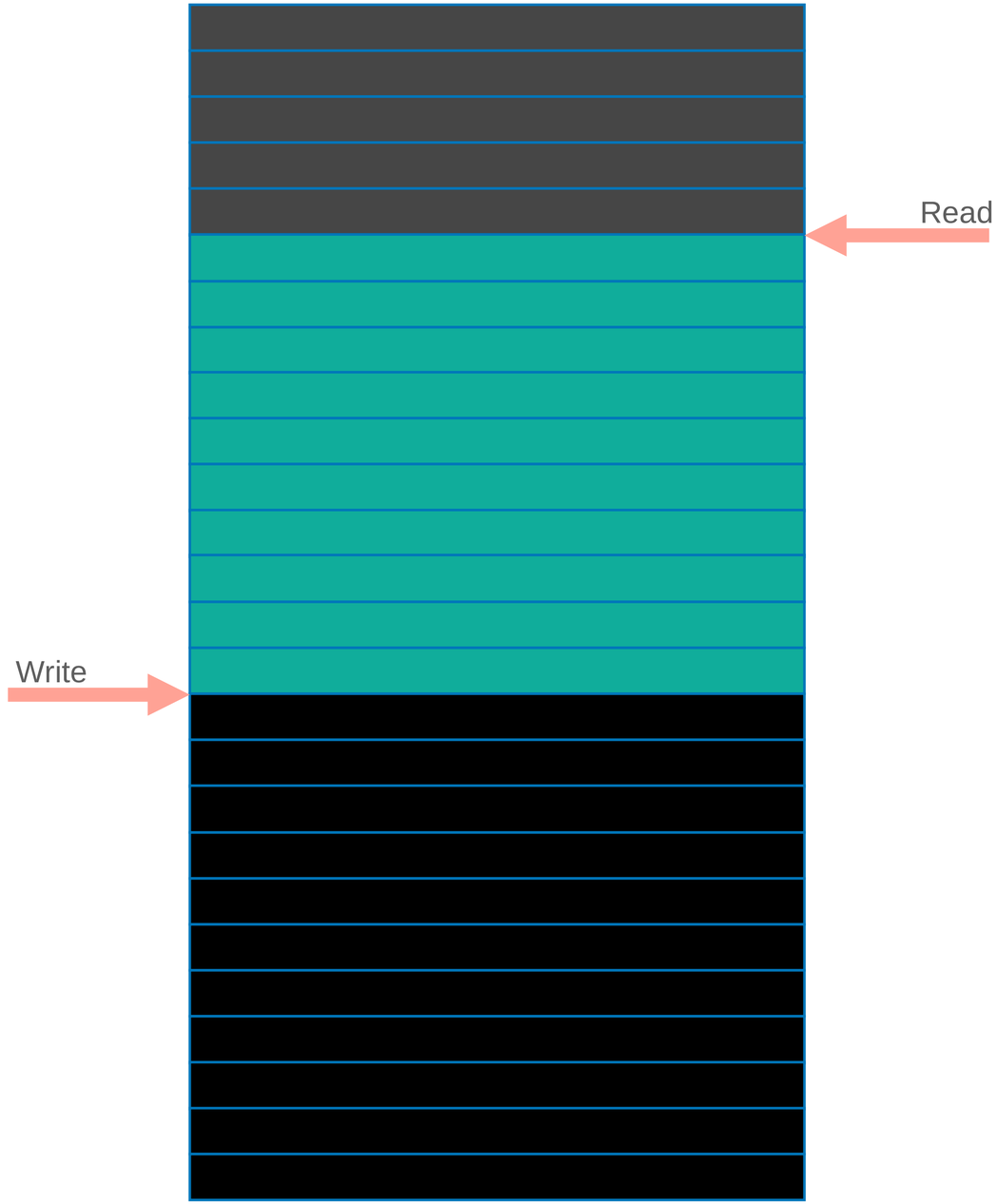


BRAM simplified schematic

Block RAM

What does this look like?

- Data for an event is written to the BRAM upon reception
- Data is read by PC as it is able to
- If write rate greatly overwhelms the read rate…

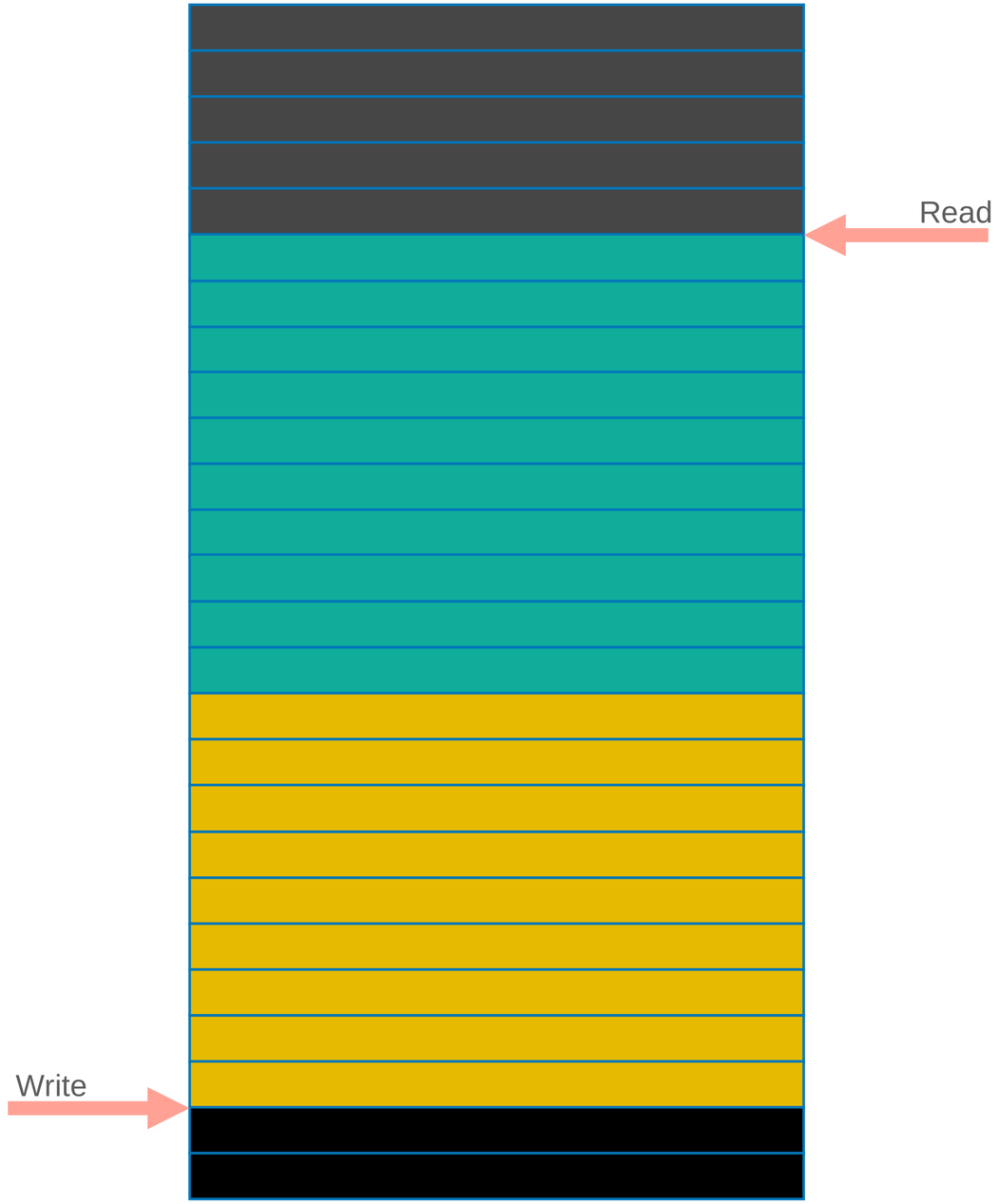


BRAM simplified schematic

Block RAM

What does this look like?

- Data for an event is written to the BRAM upon reception
- Data is read by PC as it is able to
- If write rate greatly overwhelms the read rate…

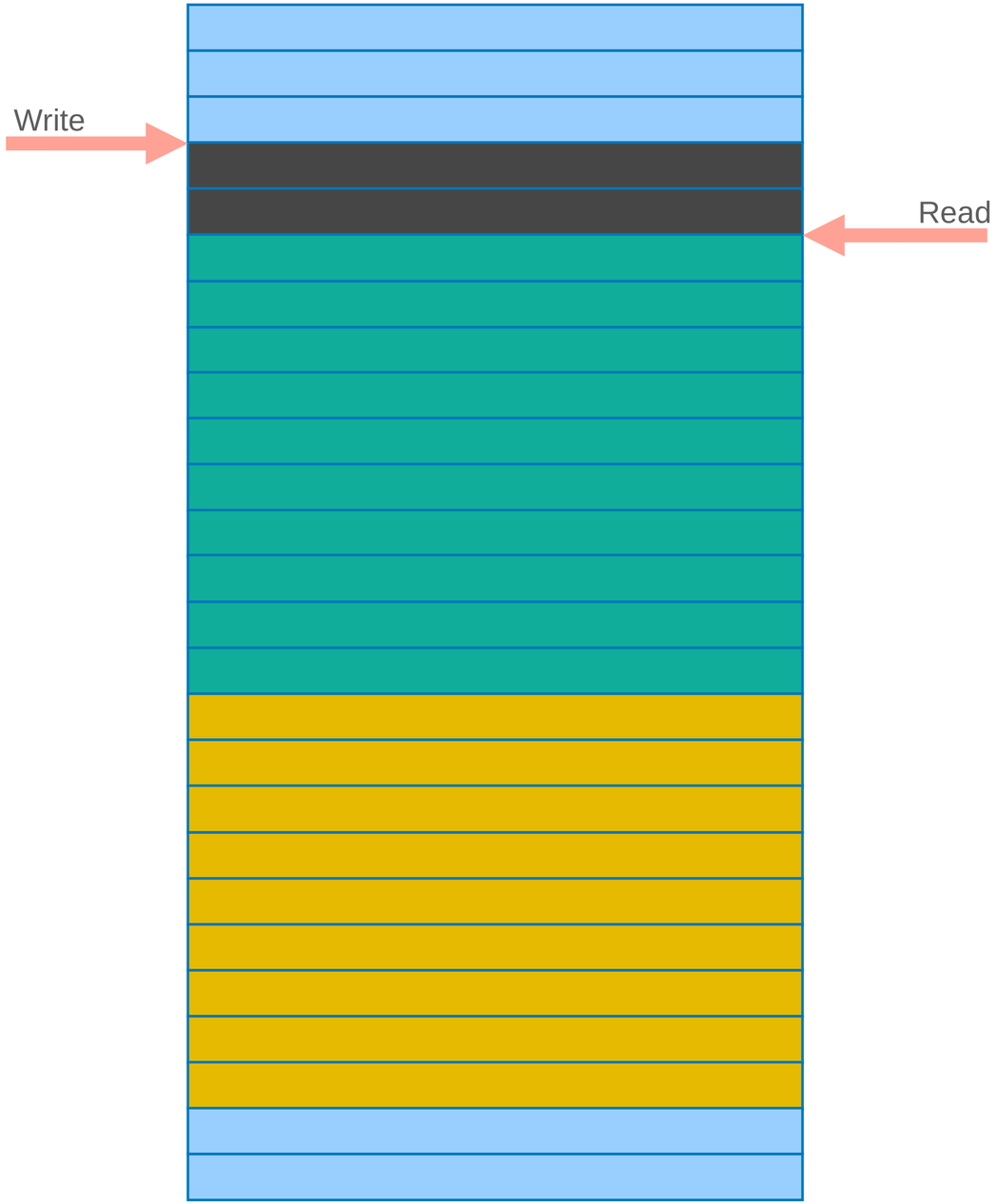


BRAM simplified schematic

Block RAM

What does this look like?

- Data for an event is written to the BRAM upon reception
- Data is read by PC as it is able to
- If write rate greatly overwhelms the read rate…



BRAM simplified schematic

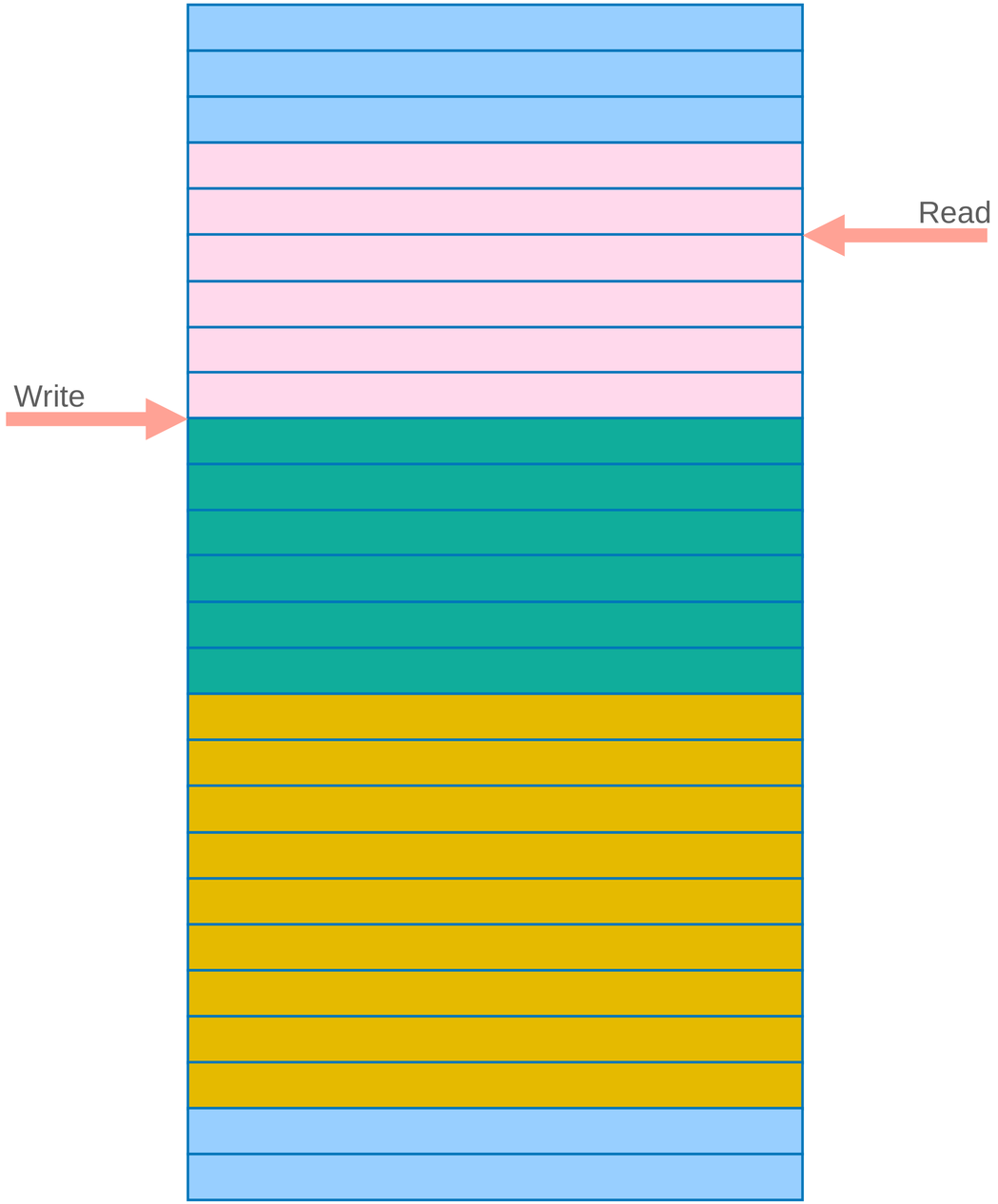
Block RAM

What does this look like?

- Data for an event is written to the BRAM upon reception
- Data is read by PC as it is able to
- If write rate greatly overwhelms the read rate…
- Data can be overwritten

In data:

- This can manifest as **dropped hits**, **unfinished event errors**, and “**expect new stream…**” data processor errors
- 64 bit chunks of data are missing from full event stream -> gives these kinds of errors



BRAM simplified schematic

More Debugging

- As we have seen, FPGA problems with data streams generally manifest as dropped 64 bit packages within a stream
  - Hits from a different module transferred from one chip to another -> BRAM overwrite
  - **Truncated** or **abruptly starting** streams -> BRAM overwrite or **bit shift** requiring resynchronization of data stream

- Chip-related data issues may have a more subtle look
  - I.e. certain core column issues: tag comes back, followed by legible streams of **garbage data**
  - In-chip memory overwrite not even visible!

- How can we try to distinguish between these issues?

More Debugging

One possible tool: **DataProcessor debug buffer**

- Keeps track of last <N> streams
- Upon reception of a data processor error, prints last N streams for that frontend
- Gives a much clearer idea of what may have happened: **full stream context**
- However, requires decoding by hand (ugh!) (get to know the **V2 manual**)

To use:

- Can be enabled with **cmake** compiler flags:

USE_ITKPIX_DEBUG_BUFFER:

0: disable completely

1: enable only for segfault-inducing cases

2: enable for all errors

ITKPIX_DEBUG_BUFFERSIZE:

N: number of consecutive streams to keep in memory

… power cycle?

PCIe protocol must be set on startup

- So, we cannot “hard reset” the entire FPGA.. only the wishbone busses
- You *can* reach FPGA configurations that require a PC reboot to fix, if you try hard enough

That being said, the vast majority of “bad” FPGA configurations can be fixed by running **specSoftReset** (mentioned earlier)

```
[ 0:49:04] luclepot@pigwhale $ ./bin/specComTest 1
[00:49:08:385][  info  ][  specComTest  ]: Init spec
[00:49:08:385][  info  ][    SpecCom    ]: Opening SPEC with id #1
[00:49:08:385][  info  ][    SpecCom    ]: Mapping BARs ...
[00:49:08:385][  info  ][    SpecCom    ]: ... Mapped BAR0 at 0x7f44b264f000 with size 1048576
[00:49:08:386][warning ][    SpecCom    ]: ... BAR4 not mapped (Mmap failed)
[00:49:08:386][  info  ][    SpecCom    ]: ~~~~~~~~~~~~~~~~~~~~~~~~~~
[00:49:08:386][  info  ][    SpecCom    ]: Firmware Hash: 0x1fe6cb
[00:49:08:386][  info  ][    SpecCom    ]: Firmware Version:
[00:49:08:386][  info  ][    SpecCom    ]: Firmware Identifier: 0x2030248
[00:49:08:386][  info  ][    SpecCom    ]: FPGA card: Trenz TEF1001_R2
[00:49:08:386][  info  ][    SpecCom    ]: FE Chip Type: RD53A/B/C
[00:49:08:386][  info  ][    SpecCom    ]: FMC Card Type: Ohio Card (Display Port)
[00:49:08:386][  info  ][    SpecCom    ]: RX Speed: 1280Mbps
[00:49:08:386][  info  ][    SpecCom    ]: Channel Configuration: 12x1 Ext Trig
[00:49:08:386][  info  ][    SpecCom    ]: LPM Status: 0
[00:49:08:386][  info  ][    SpecCom    ]: ~~~~~~~~~~~~~~~~~~~~~~~~~~
[00:49:08:386][  info  ][    SpecCom    ]: Flushing buffers ...
[00:49:08:386][  info  ][    SpecCom    ]: Init success!
[00:49:08:386][  info  ][  specComTest  ]: Starting DMA write/read test ...
[00:49:08:386][  info  ][  specComTest  ]: ... writing 8192 byte.
[00:49:08:386][  info  ][  specComTest  ]: ... read 8192 byte.
[00:49:08:386][  info  ][  specComTest  ]: Success! No errors.
```

Works!

```
[ 0:50:28] luclepot@pigwhale $ ./bin/specComTest 0
[00:50:36:818][  info  ][  specComTest  ]: Init spec
[00:50:36:819][  info  ][    SpecCom    ]: Opening SPEC with id #0
[00:50:36:819][critical][    SpecCom    ]: Error during initilisation:  Lock failed
[00:50:36:819][critical][    SpecCom    ]: Fatal Error! Aborting!
```

Broken!

Questions