Pix2PGP

ASIC Implementation Architecture

Christos Bakalis
TID - TID-ID-ECS

15 May 2024

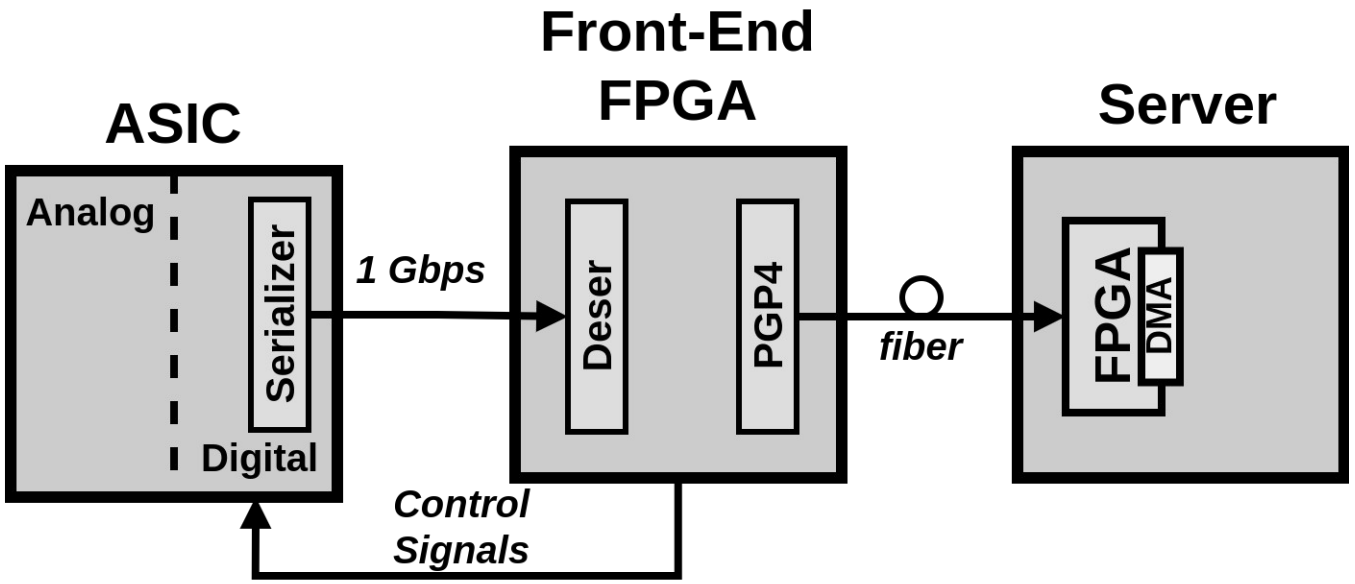
SLAC NATIONAL ACCELERATOR LABORATORY
Stanford University | U.S. DEPARTMENT OF ENERGY

Outline

- Introduction
  - Current Sparisifed Readout Analog/Digital Architecture
- Next Step – Pix2PGP – System Overview
- Current Architecture of ASIC Data Extraction
- Proposed ASIC Architecture - Pix2PGP
  - Column Manager
  - Arbiter and Column Supervisor
  - Gearbox and Data Structure Considerations
- Summary
- Bonus at Backup: Error Handling and Overhead/Bandwidth Calculations

Introduction

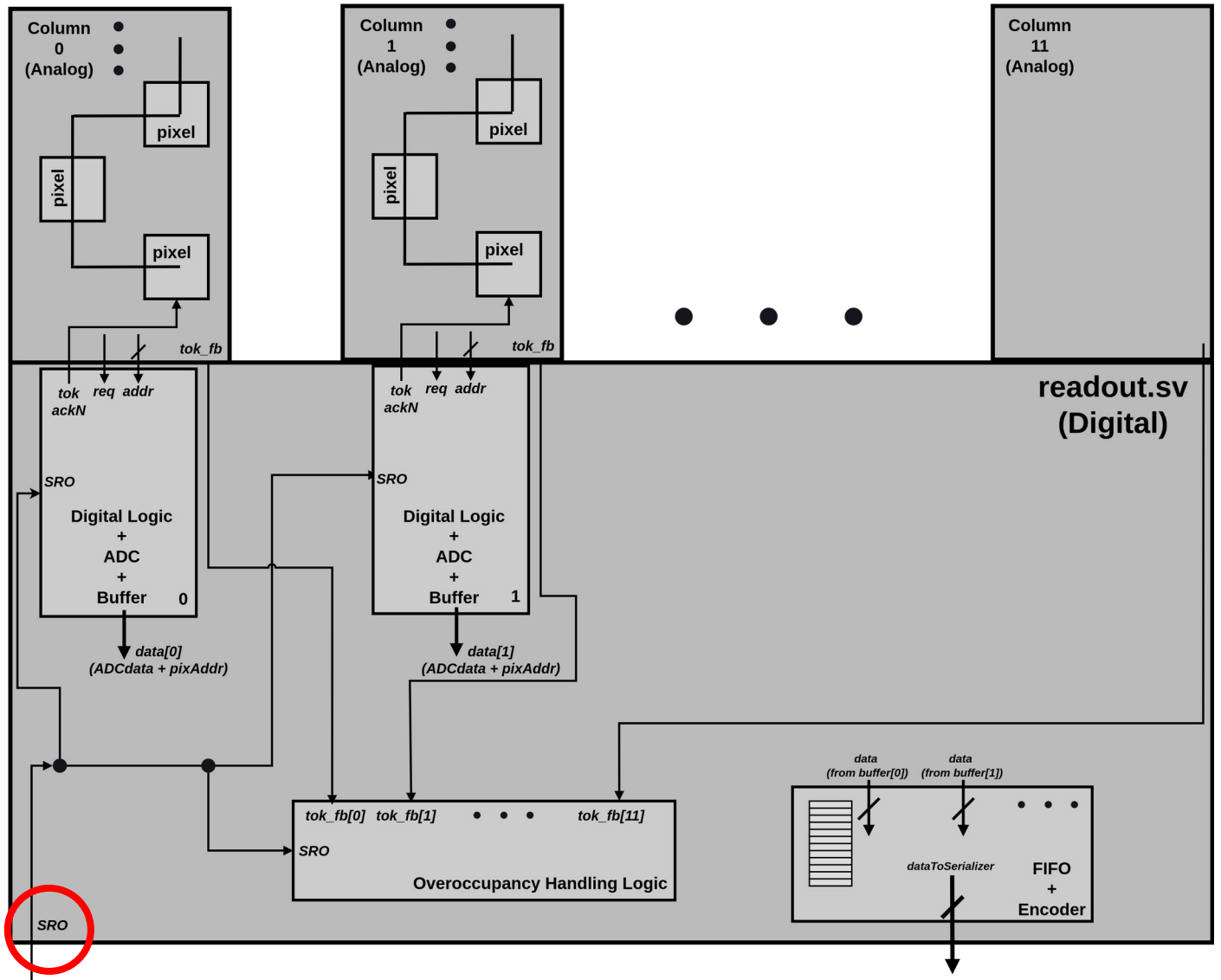
Currently, the data handling for Sparsified Readout ASICs (SparkPix-T and -S) support limited bandwidth



Current Sparsified Readout Digital/Analog Architecture

SRO is driven by Front-End FPGA.

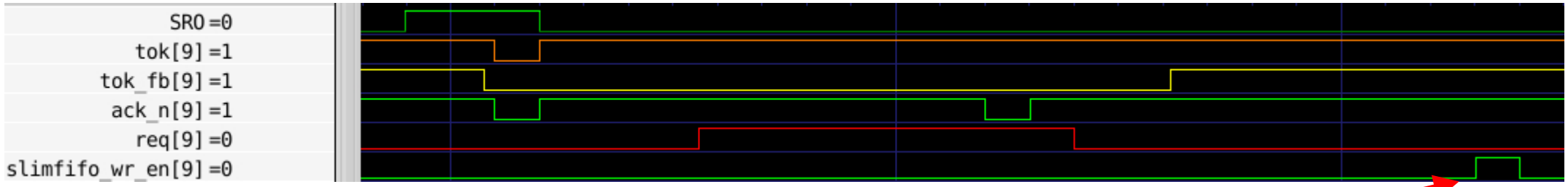
SRO is essentially the main trigger that initiates the readout of pixels



Each pixel column accepts the *SRO* in the form of a token *(tko)*. If a pixel has data, it requests to be digitized *(req)*. Data are then converted and buffered

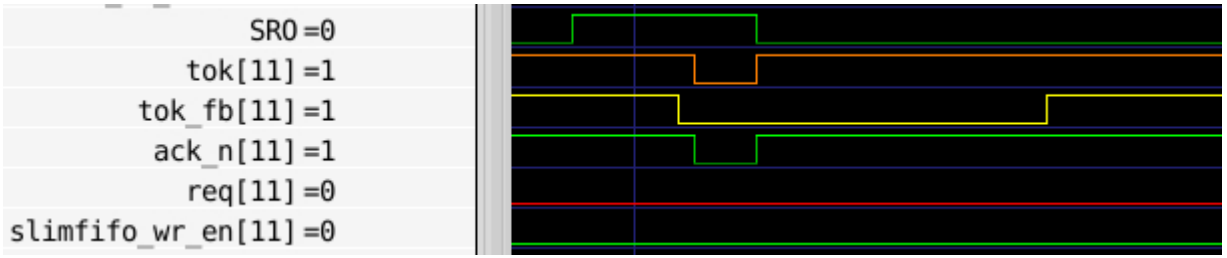
Example Waveforms

One pixel of column[9] has data



The ADC conversion process induces a latency → *wrEn* for the data word (ADC word + addr is issued 'late')

No Data on column[11]

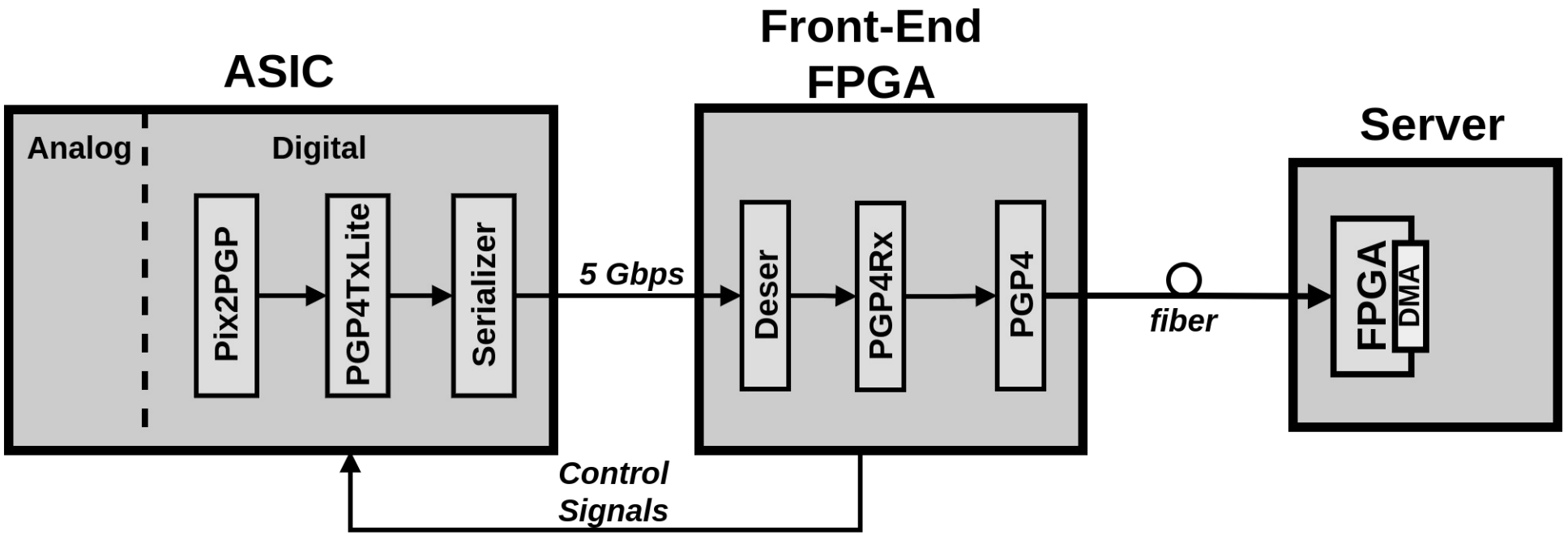


Overoccupancy **Error** → receive new *SRO* before *tok_fb* goes high
OR
receive new *SRO* before last *FIFOwrEn* is issued

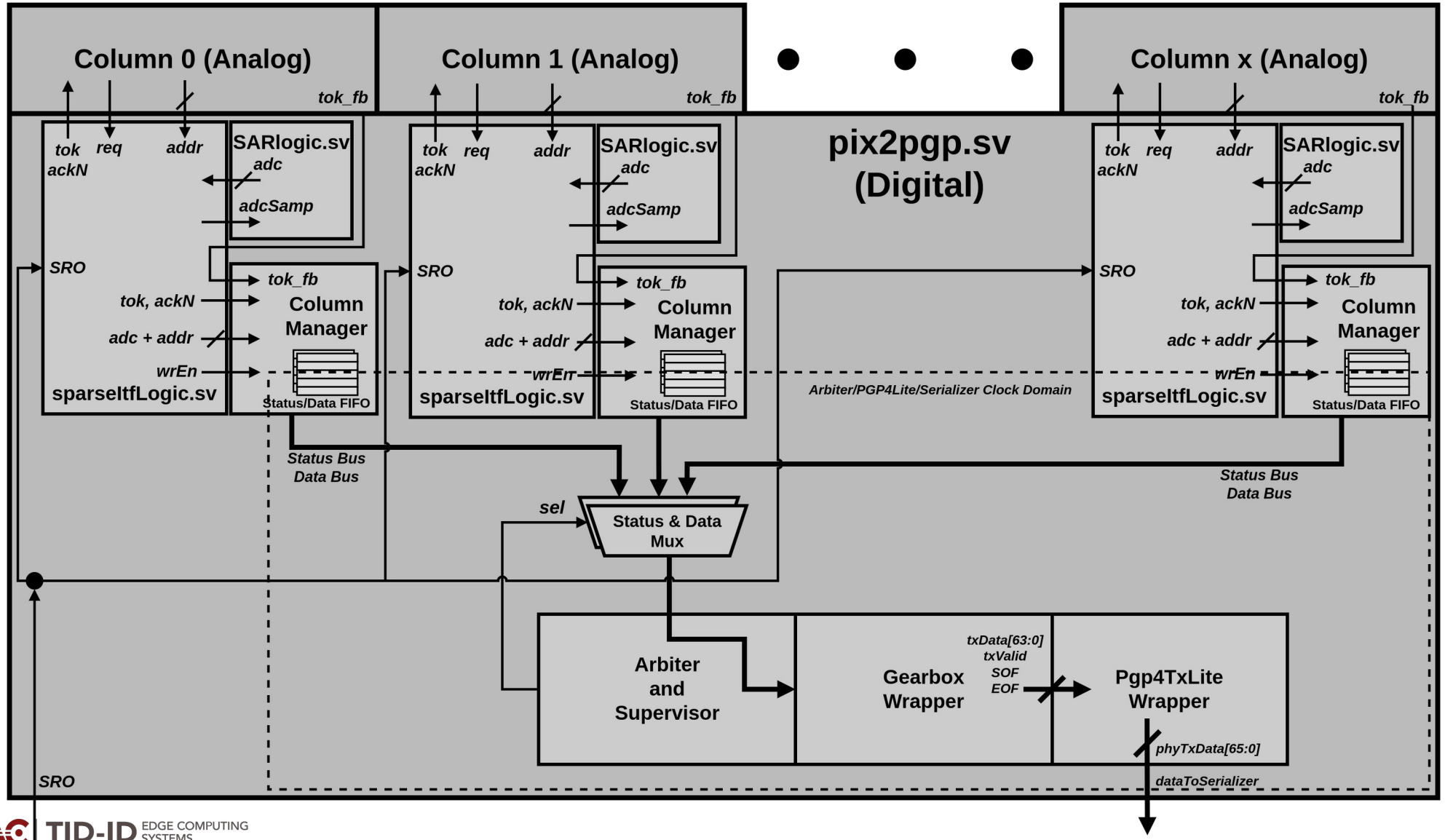
Pix2PGP – System Overview

- Need to **increase** the bandwidth to comply with **higher** trigger rates of LCLS-II
- First steps towards that include the deployment of a **new ASIC serializer** in the ASIC, driven by the Pgp4TxLite surf module.
  - Pgp4TxLite is a 64B/66B encoder that implements the PGP4 Protocol with minimal gate utilization → ideal for ASICs
- The functionality of the **new ASIC serializer** has been demonstrated **successfully** through the SparkPix-IO project
  - Pgp4TxLite not yet tested on the ASIC level (to-be-tested soon on the ePixUHR project)

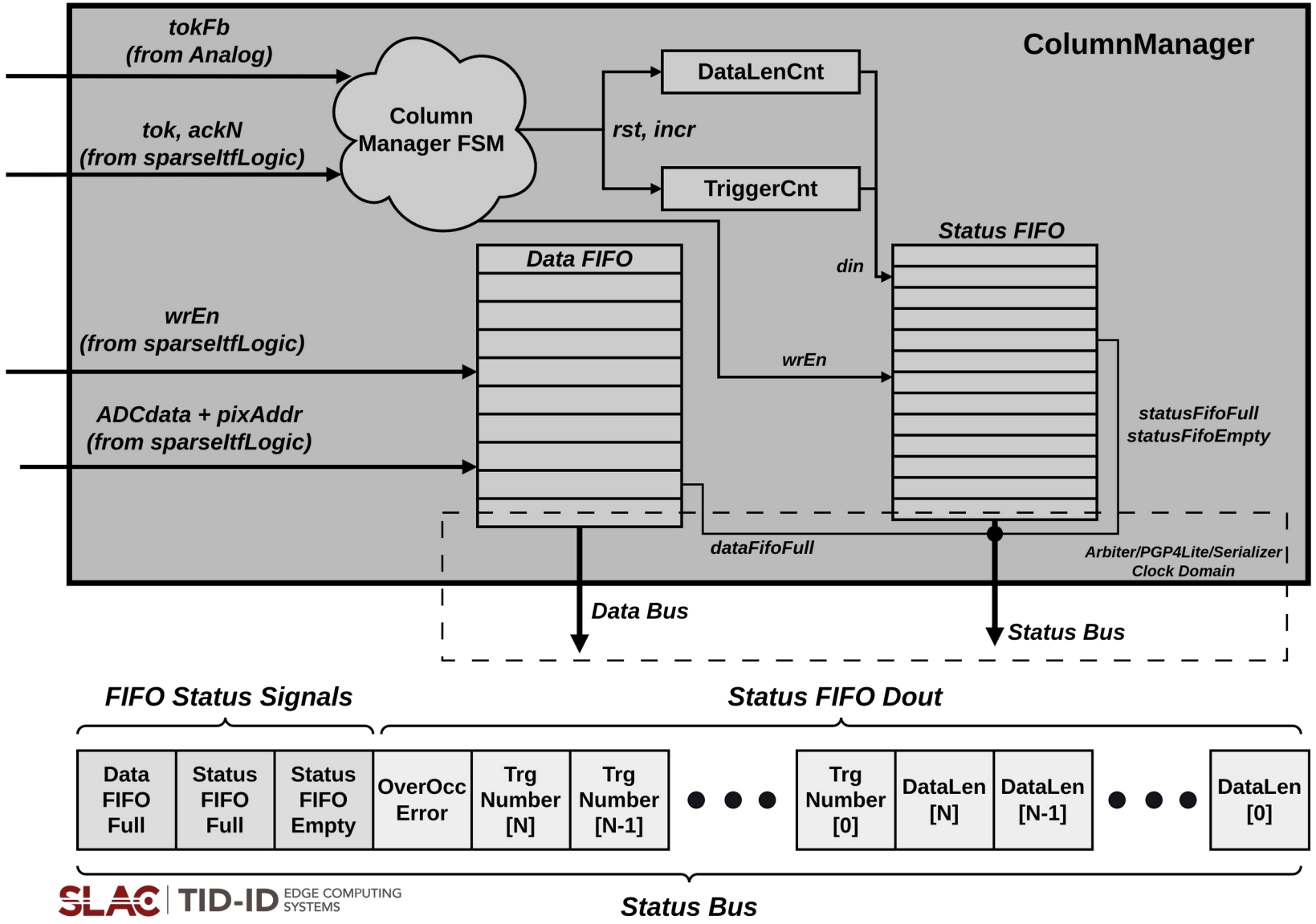
System Overview – Pix2PGP and PGP4TxLite



Proposed Architecture – Pix2PGP Top-Level

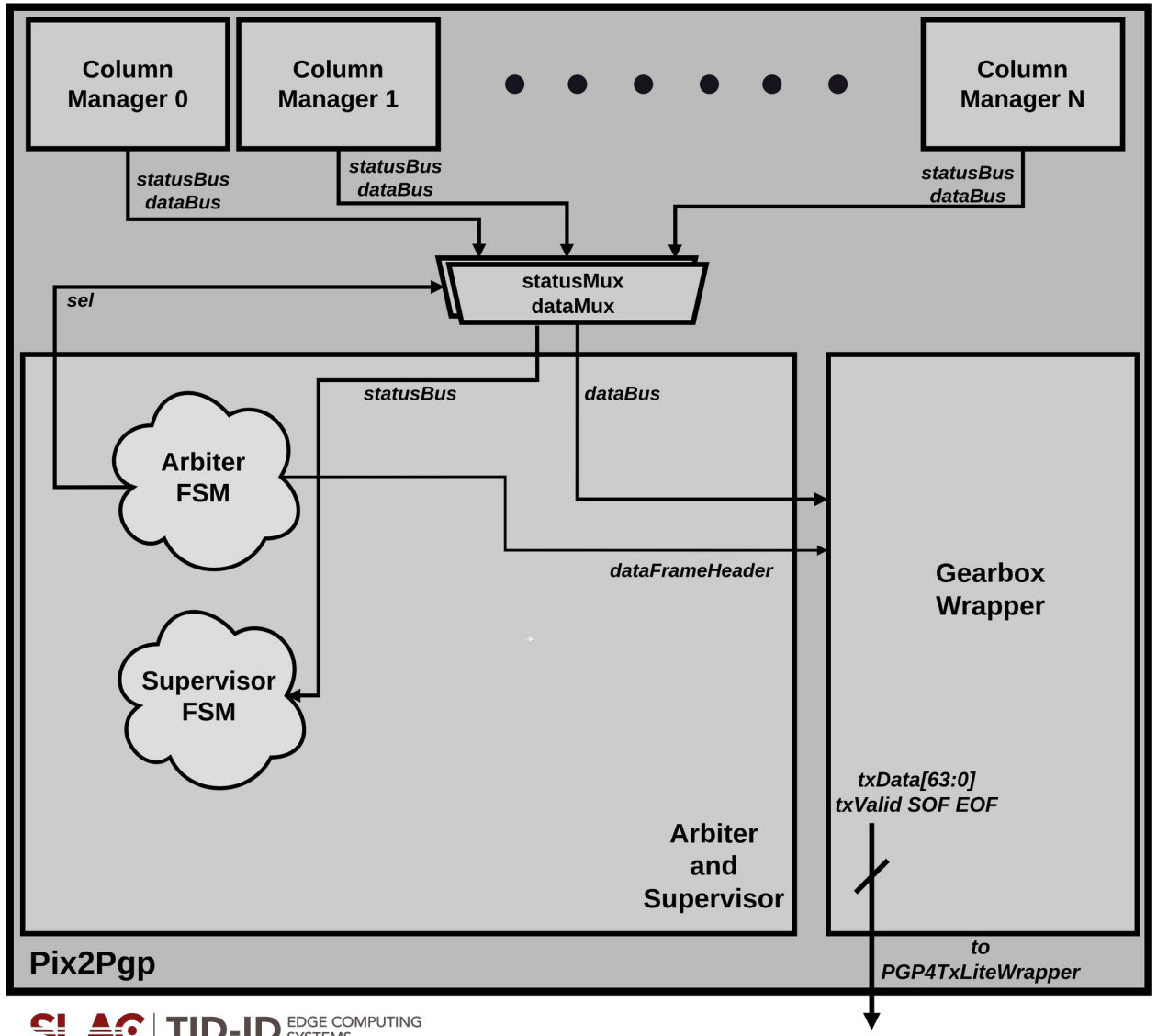


Column Manager



- Trigger Counter increments for each received *token/SRO*
- FSM waits for request to convert *(req)*. If one request is received, DataLenCnt += 1. If two requests, DataLenCnt +=2 etc.
- The converted ADC data + pixAddr are written straight to the Data FIFO
- Upon reception of *tok_fb* (i.e. when analog is **done** scanning pixels for hits), the FSM writes the the associated **status word** to the Status FIFO
- If overoccupancy → the FSM raises the OverOcc bit and writes into the Status FIFO

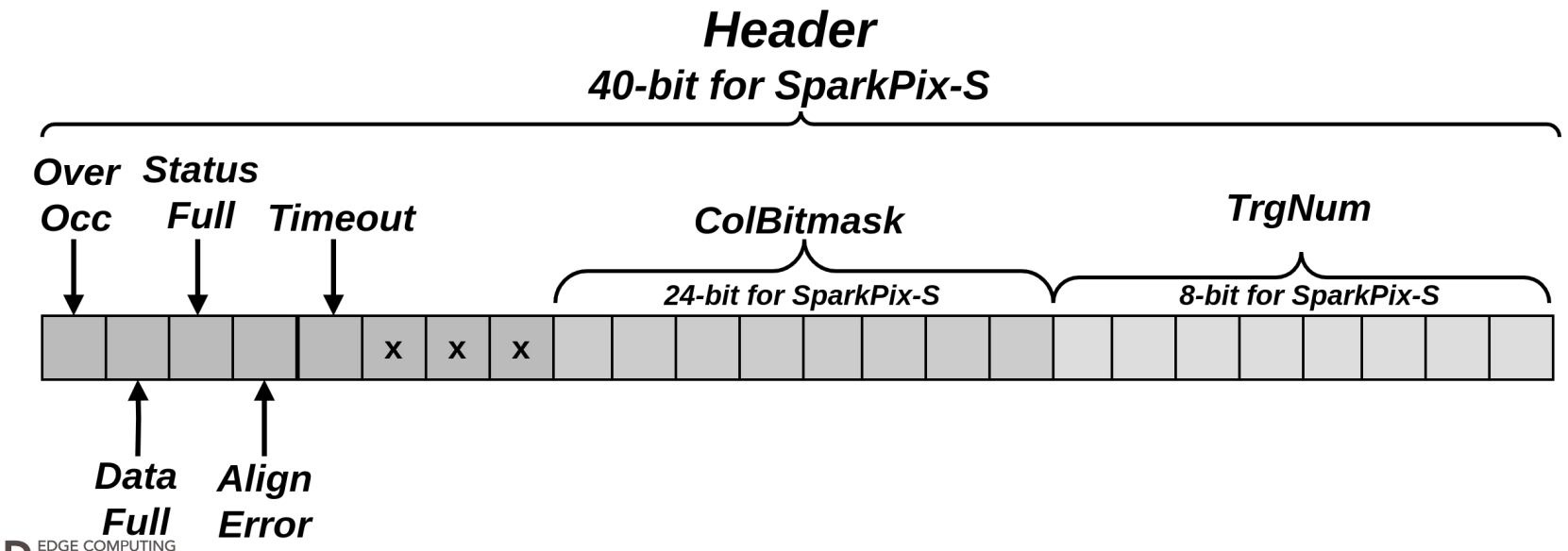
Arbiter and Supervisor



- The Supervisory process monitors the *StatusFifoEmpty* signal of **all** status FIFOs
  - If **all** Status FIFOs have a word, that word (xN) is read and evaluated on a global scale
    - If some FIFOs are full, an error data frame header is TX'd (same for other errors)
    - If no errors but also no Column has data, a data Frame header is forwarded to the Gearbox wrapper
    - If some (or all) Columns have data, the Arbiter round-robins and forwards the data to the Gearbox wrapper, alongside the data Frame header

Data Frame Structure (for SparkPix-S)

- The Pix2Pgp data frames always consist of one 40-bit header
  - 8-bit flag Field (mostly Error bits)
  - 24-bit Column Bitmask Field (if a bit high → corresponding column has data)
  - 8-bit Trigger ID (from the Column Status FIFO – should be the same for all Columns)



Data Frame Structure (for SparkPix-S)

- If no hits on any column, only the header is TX'd. Same if there is an error (except for OverOcc error, where data are still read-out)
  - If **Error**, the receiver FPGA will register that and probably issue a **reset** and notify the software
- If data are extracted for a given event/trigger, then each Column with hits will have **one** 10-bit dataLen word (from its Status FIFO) and the amount of 20-bit hits it recorded.
  - E.g.: this event has 2 hits from two different cols (cols 0 and 5)
    pgp data frame header | *col5_dataLen=1* | *col5_hit0* | *col0_dataLen=1* | *col0_hit0*
  - E.g. 2: this event has 3 hits from one column (col 2)
    pgp data frame header | *col2_dataLen=3* | *col2_hit0* | *col2_hit1* | *col2_hit2*

See backup for expected data/frame length calculations

Frame size and Gearbox considerations

- Since PGP4Tx works with 64-bit words, it is convenient if its input words are 64-bit wide
- Each PGP data frame for SparkPix-S consists of:
  - One 40-bit header (100% certain)
  - Max 24x10-bit dataLen words (if data in some or all columns)
  - Max 24x672x20-bit hit words (if data in some or all columns)
- So since all of these elements are multiples of 10 bits, this Gearbox configuration was chosen:
  - **10-bit → 320-bit → 64-bit***
  - The **first** gearbox will send **one** 320-bit word to the **second** gearbox, per 32x10-bit words as generated by the Arbiter.
  - Then the **second** gearbox will write **5x64-bit** words into an AXI-Stream FIFO → PGP4Tx Input

**Seems odd? It will become more clear why on the next slides*

Frame size and Gearbox considerations

This Gearbox configuration will allow to “flush” it very easily after receiving the last trigger of a given sequence

- The gearbox will probably have some data remnants in its shift-register if data are not being pushed out of the columns anymore
  - i.e. if no triggers are issued anymore, there should be some data associated with the last trigger left in the 10:320 gearbox shift-register
- By querying the gearbox data pointer position, the Pix2PGP logic can insert dummy timeout headers (recognized by the receiver from its associated flag) and push the last valid data out of the pipeline. This is the main reason why the 10:320:64 gearbox configuration was chosen → it allows for easy padding in that corner case
- Finally, we can configure via SACI on how large we want the PGP frames to be. The default can be equal to one burst of five 64-bit words; but we can increase that accordingly. It all depends on how large an event we expect → this will decrease the PGP protocol overhead*

*See backup for expected data/frame length calculations

Frame size and Gearbox considerations

Therefore, the PGP frame delimiters will ***not*** be the same as the event delimiters

- Due to the frame structure though, the FPGA receiver **will be able to process** the inbound PGP frames from the ASIC, and separate the events
  - It essentially knows how long the event frame within one or multiple PGP data frames will be and will break down the events from within the PGP frames
  - It will then operate as it operates now, where it sends ONE PGP4 frame to the back-end software per ONE event. This will also allow for aligning the data with the timing information from the LCLS2 Timing Receiver

Conclusion

- The proposed architecture allows for reading-out columns that will be operating **independently**
  - The concept of writing the length of the data alongside the data themselves into FIFOs allows for **fast** and **independent** operation between Analog and Digital
  - The presence of the trigger counter and the overall frame structure allows for **easy event reconstruction** by the FPGA receiver
  - The proposed **gearbox structure facilitates** the driving of the PGP interface, handling of overall PGP frame size and resolution of corner-cases that arise when no more triggers are being received by the ASIC
- Each column handles a possible **overoccupancy** situation independently
  - A standardized frame is TX'd via PGP for each **error** case, providing system-level updates
    - The FPGA acts accordingly by resetting
- Tried to develop this **as generic as possible**. Most of the logic will be **parametrized**, in order to port it easily into more ASICs in the future. The only thing that will change will be the frame size, but the readout concept will stay the same

Status

- Most logic has been designed and can be analyzed with GHDL
- Some testbenches are available. Working on making more and putting it all together

https://github.com/slaclab/pix2pgp

Backup

Error Handling on the FPGA Receiver – Error Flags

- **Over-Occupancy** (flag controlled by the Column Manager)
  - Occurs if an SRO is received by the Column Manager/Analog during processing of a previous event (i.e. before the token-feedback is issued after the SRO).
  - The ASIC still TX's the data alongside the OverOcc flag in the header
  - The FPGA receiver will receive the data, and also register the OverOcc flag and forward both to the receiver software. TBD: Reset the ASIC? Not Reset?
- **Trigger Misalignment** (flag controlled by the Column Supervisor)
  - Occurs if a Column Manager has a different trigger value with respect to the rest during the same event. The FPGA receiver will discard the received data and reset the ASIC. It will also report the error to the receiver software
- **Status/Data FIFO Full** (flag controlled by the Column Supervisor)
  - The FPGA receiver will reset the ASIC and report the error to the receiver software

Error Handling on the FPGA Receiver – Data Corruption

- Should not be too big of an issue since we are talking about X-Ray detectors here. In any case, if any bitflips occur, the severity of the issue will depend on *where within the frame* the bitflip is.
  - If the bitflip occurs within the **data** themselves (i.e. in a column address or ADC value):
    - **Small** impact
  - If the bitflip occurs on a **trigger ID**, it might cause misalignment between data from different serializers within the same ASIC (because the FPGA receiver aligns the data from different serializers based on their trigger ID), or between multiple ASICs
    - **Significant** impact. Need to **reset** the full chain
- If the bitflip occurs within the **colBitmask** of the header or the **dataLen** inside the data:
    - **Significant** impact. This will ruin the data handling on the receiver side, as it will mess up with the limits of the frame (the FPGA receiver will erroneously think that the event is smaller or larger)
    - The **receiver software can have safeguards** on this: Event size too big? Likely bitflip. OR, did we just receive two hits for one pixel for one event? Likely bitflip → Signal a full chain **reset**

Overhead and Bandwidth for Occupancy of 0.5% at 1MHz

3.36 hits per column (each hit word = 20 bits)

- 3.36 x 24 → ~80 hits per serializer. 80x20 ~=1600 bits of pure data
- 24x10 → 240 bits of dataLen (if all columns have at least one hit)
- 1x40 → 40 bits for the header

- 1600 (effective user bits) + 240 (overhead) + 40 (overhead) = 1880 bits per trigger
  - Therefore, Pix2PGP data frame overhead is ~15%
  - If we configure the logic between Gearbox and PGP for a frame size of 1920 bits → One 30x64-bit PGP frame will contain roughly one event (i.e. 6x(5x64)-bit gearbox bursts)
  - PGP adds one SOF (64 bits) and one EOF (64 bits) to each frame it TX's (CRC is in the EOF)
  - PGP adds 2 control bits per 64-bit word → a PGP frame of 30x64-bit words will have an overhead of 2x30 + 64+2 + 64+2 → 192 bits overhead from PGP4 (1% overhead for the aforementioned data frame size)
    - 1880 (Pix2PGP frame) + 192 (PGP4 Overhead) ~= 2100 bits/s → **2.1 Gbps** (at 1MHz)

Well below the 5 Gbps serializer bandwidth