
# Zynq US+ MPSoC in the gFEX Hardware Trigger in ATLAS

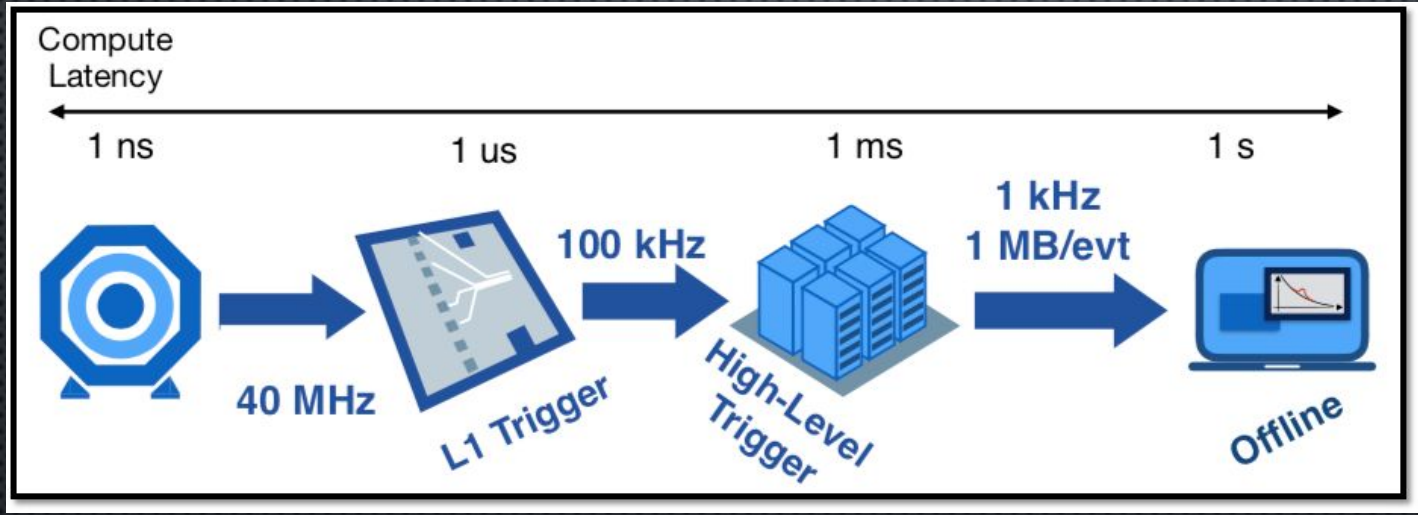
June 8th, 2021
SoC Workshop
Emily Smith
gFEX Team


THE UNIVERSITY OF CHICAGO

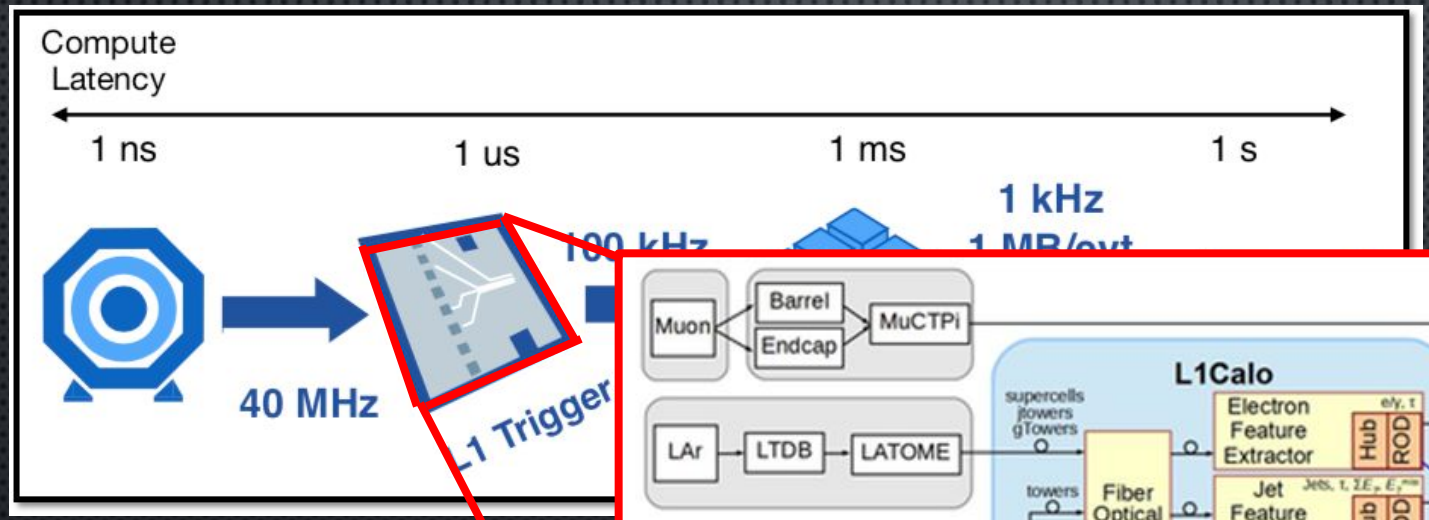
# Outline

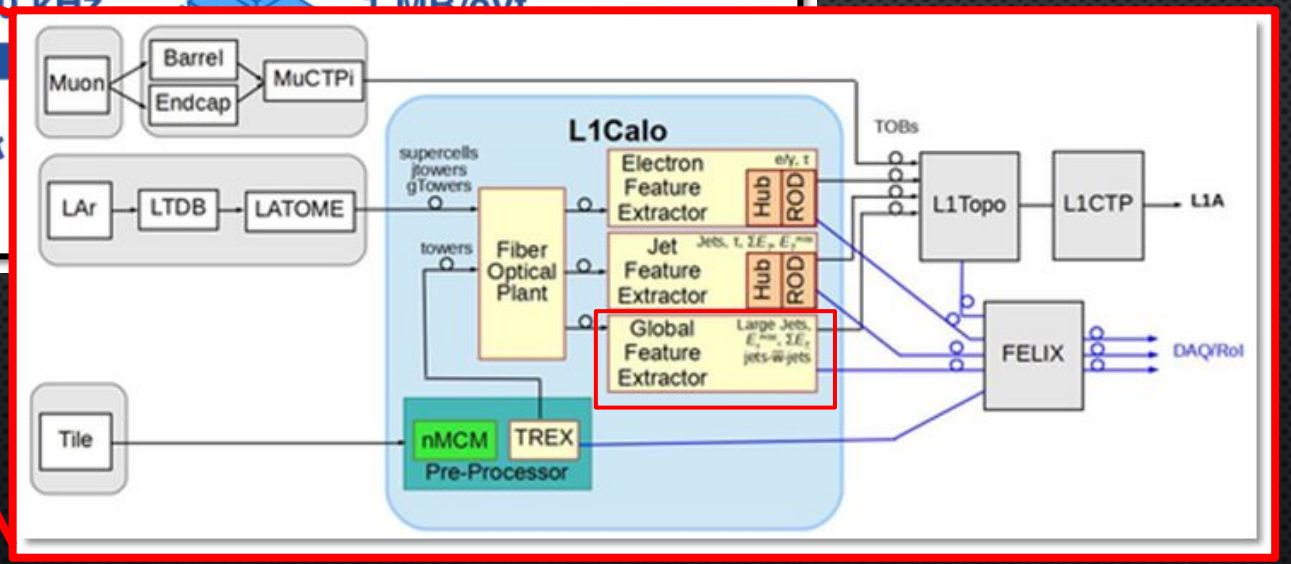
- gFEX (global Feature EXtractor) Introduction
- Operating System Workflow
- Zynq SoC Monitoring and Control
- Processing System (PS) / Programmable Logic (PL) Interfaces (PYNQ)

# ATLAS Hardware Calorimeter Trigger for LHC Run 3



# ATLAS Hardware Calorimeter Trigger for LHC Run 3

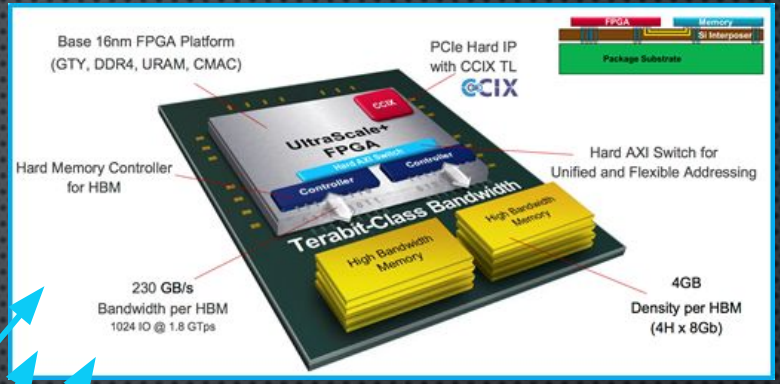


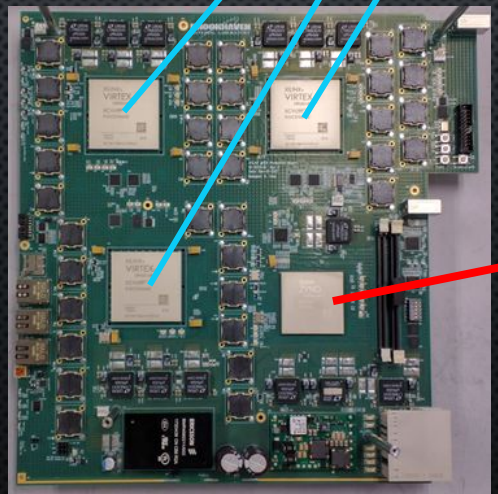



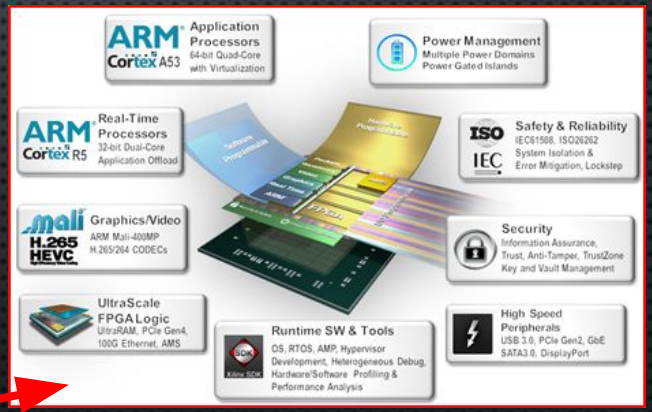
# gFEX Hardware Design

- Entire calorimeter read out on one board for calculating global observables such as large-radius jets, MET, and pile-up estimation for triggers
- 3 Ultrascale+ (VU9P) processor FPGAs
- Zynq Ultrascale+ MPSoC (ZU19EG) for control and additional processes

Virtex Ultrascale+

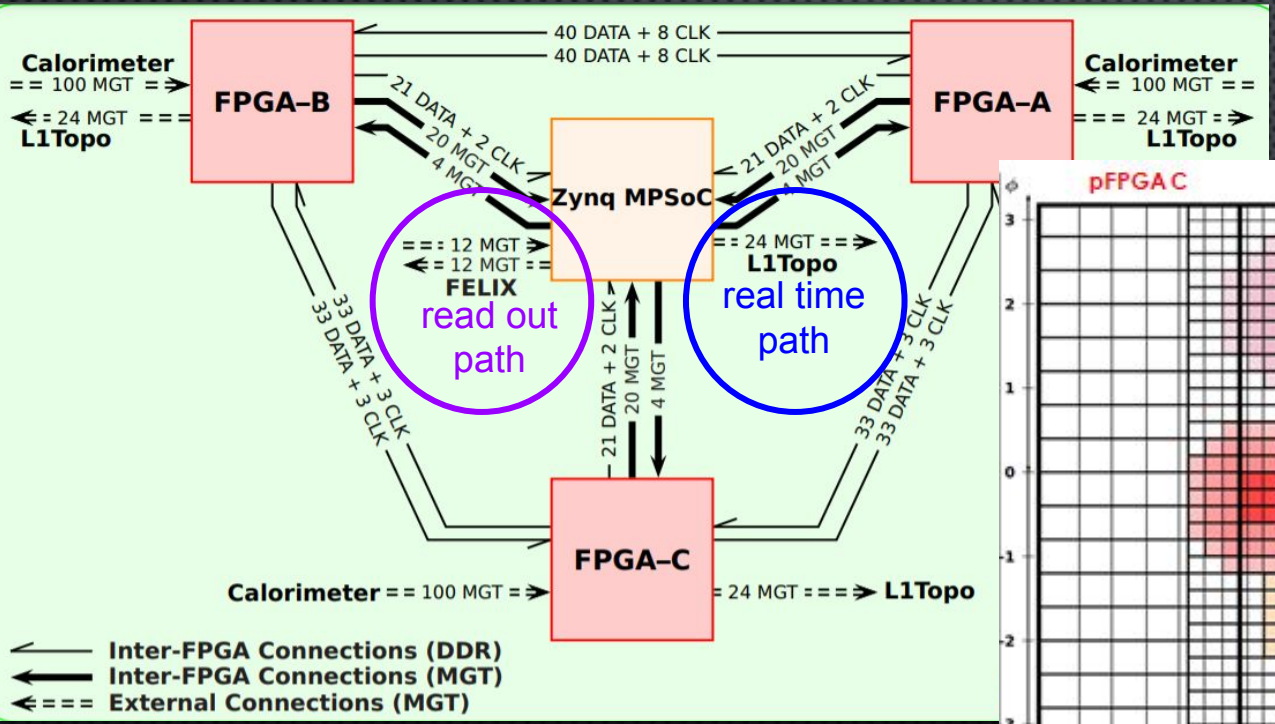


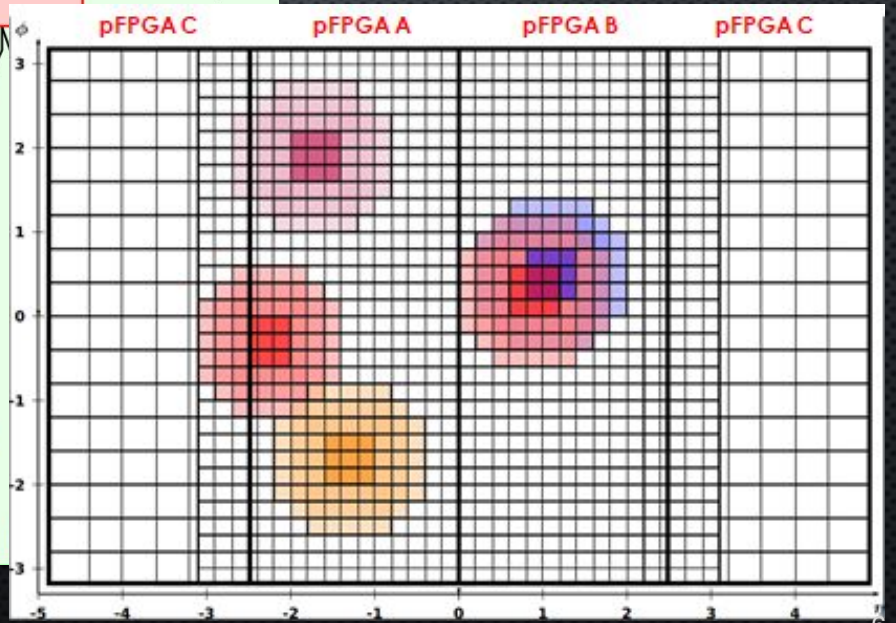






Zynq+ MPSoC

# gFEX Zynq SoC Usage

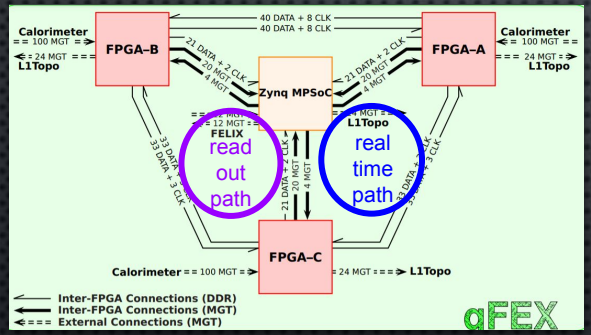


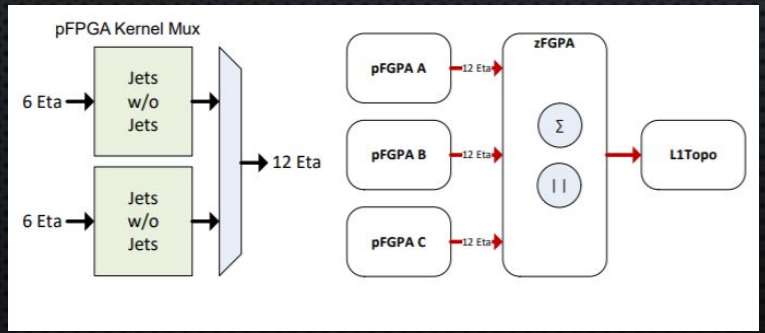



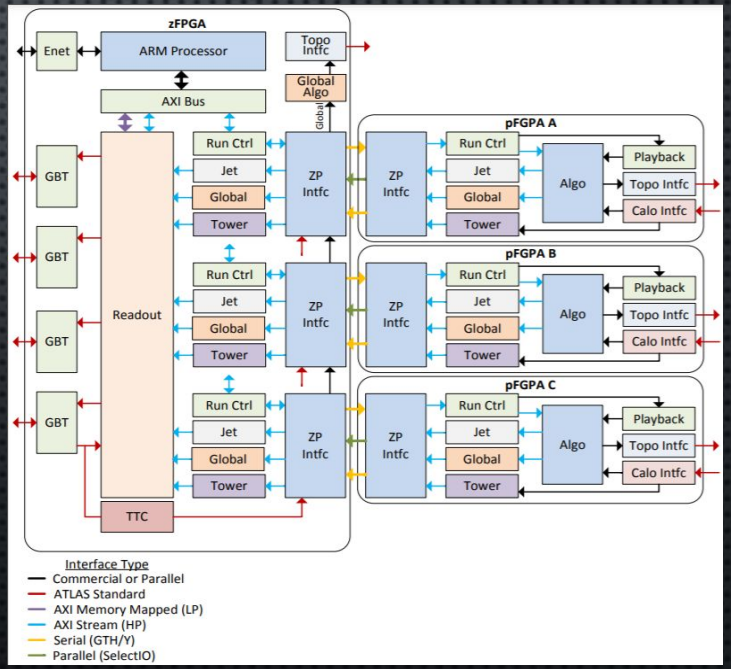
# Zynq US+ MPSoC in the real time path

pFPGA ⇄ zFPGA connection is GTH/GTY
UltraScale+ GTH (16.3Gb/s)
UltraScale+ GTY (32.75Gb/s)

- Zynq also provides real-time data processing functionality for global MET TOBs



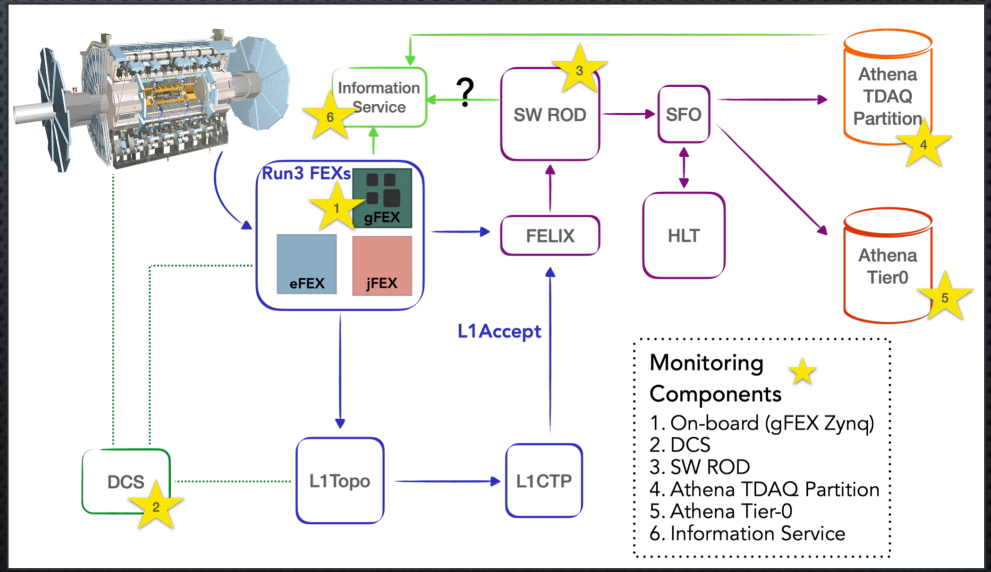
# gFEX Custom Operating System

- Custom OS developed with Yocto/Openembedded build engine and bitbake
- meta-l1calo layer is where the “custom” design is done (link)
- Originally used open-source yocto releases (rocko, sumo, zeus etc) but have now switched to slightly less open source yocto-manifests from Xilinx (rel-v2020.1, rel-v2020.2)
- Yocto-manifests benefits:
  - easier firmware integration: can pass in XSA file from firmware development to OS build
  - easier updating to newest Vivado
  - boot files built with OS (instead of externally with Vivado)
- CI on OS build in meta-l1calo repo
- Looking into the addition of the XVC (Xilinx Virtual Cable)
- Plans to investigate CentOS as well when time

| Xilinx Release Branch | Yocto Codename | Yocto Release | Linux Kernel |
|---|---|---|---|
| rel-v2021.1 | Gatesgarth | 3.2 | 5.10 |
| rel-v2020.1<br>rel-v2020.2 | Zeus | 3.0 | 5.4 |
| rel-v2019.2<br>rel-v2019.1 | Thud | 2.6 | 4.19 |
| rel-v2018.3<br>rel-v2018.2<br>rel-v2018.1 | Rocko | 2.4 | 4.14 |

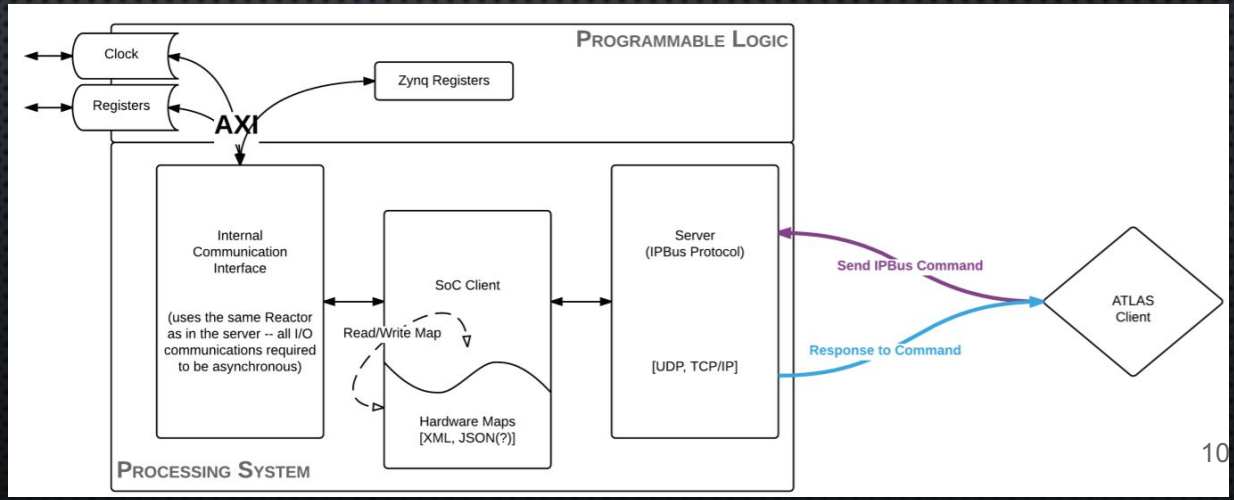
# gFEX Zynq SoC: Monitoring and Control

- Zynq primary task is monitoring and control from L1Calo central software and monitoring from DCS, also interacts with IPMC (Intelligent Platform Management Controller)
  - via IPBus packets
  - CERN v4 IPMC monitoring
  - DCS hardware sensor monitoring
- Zynq also receives algorithm output, does minimal processing, and then sends data downstream
  - Offers potential for additional processing here at a lower rate than that required by the real-time path (trigger path)

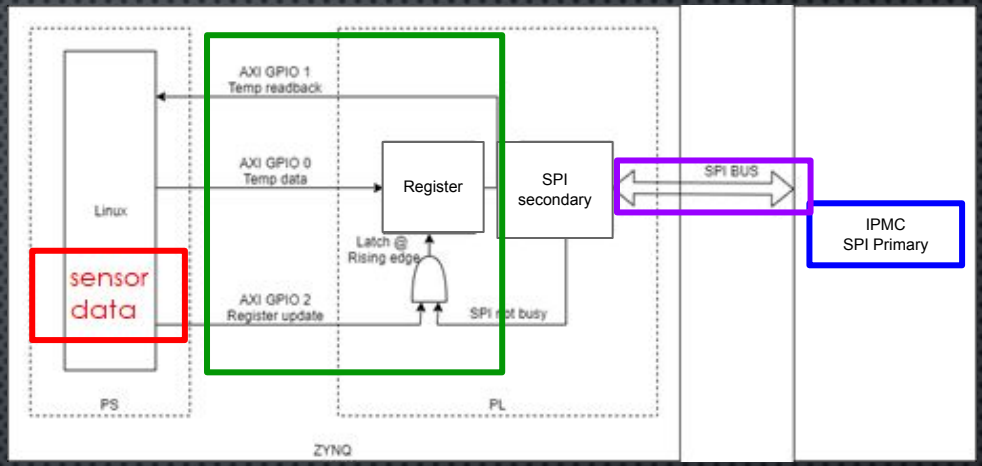


# IPBus Control and Monitoring: Ironman

- One of the greatest advantages of the Zynq SoC: performs many functions that were previously required by the firmware, but are much easier to handle in software
- Using custom python module Ironman we can receive and process IPBus packets, and send response packets
  - Extremely flexible, can be used in many scenarios
  - Much easier to implement than a firmware based solution



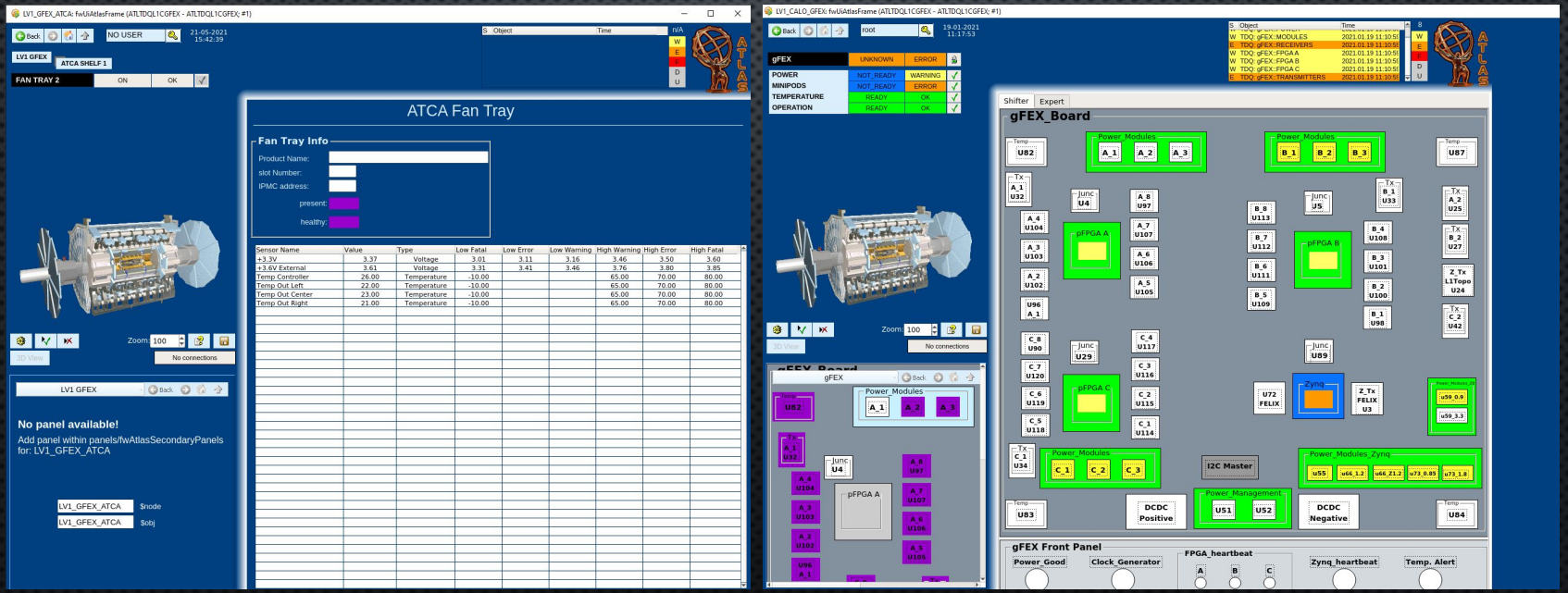
# gFEX IPMC - CERN v4




- IPMC = Intelligent Platform Management Controller
- Interface between board & ATCA shelf it's installed in
- Updating from the LAPP IPMC → CERN IPMC
- Temperature sensors on the board are read, the maximums found for different sensor types, and these values sent to the programmable logic (PL) using the GPIO. Then the IPMC can access the GPIO over the SPI Bus.
- Required an OS update with a new device tree entry for the GPIO

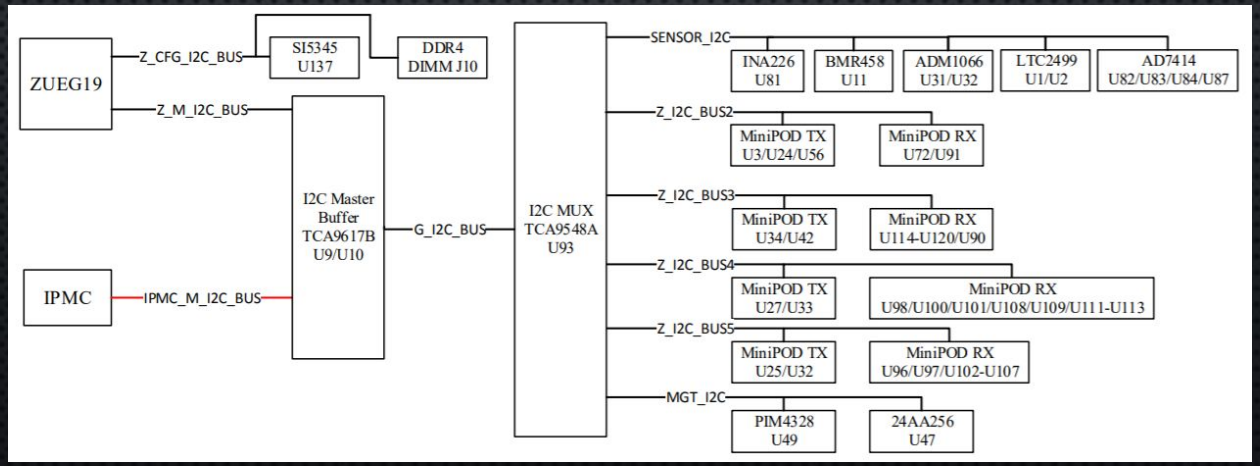
# gFEX Detector Control Systems (DCS) Monitoring

- On-board OPC UA server generated with a quasar framework (link)
- Developed and integrated into the Yocto OS build



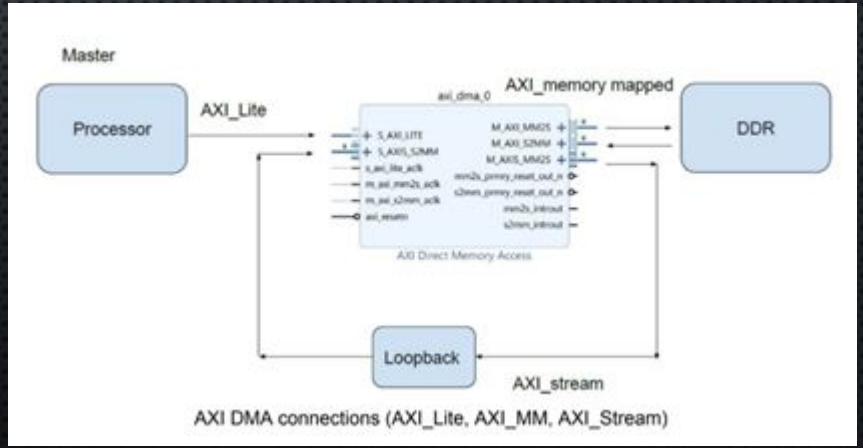
# gFEX Detector Control Systems (DCS) Monitoring

- Imported into python using the quasar Poverty python module
- Sensor values accessed over I2C using the [periphery python module]
- A script which runs on boot will query the I2C for each sensor we're interested in and write the results to a .json file, such that other processes can access the value there whenever needed (necessitated by serial access nature of I2C).



# Zynq SoC PS - PL Interfaces

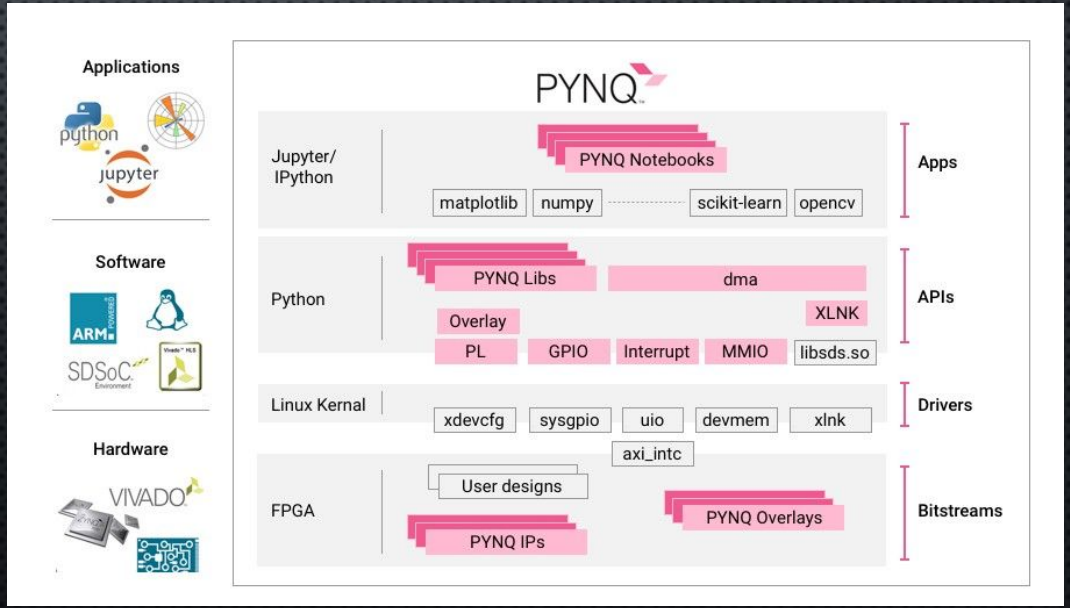
- Have used AXI-FIFO interfaces between Zynq FPGA and other FPGAs on gFEX
- Moving towards the use of AXI-DMA, "AXI DMA provides high-bandwidth direct memory access between memory and AXI4-Stream target peripherals"
- Much quicker transfer of data between FPGAs and between PS and PL on the Zynq
- Implementing the software driver side using PYNQ



AXI DMA connections (AXI_Lite, AXI_MM, AXI_Stream)

# PYNQ with the Zynq MPSoC

- PYNQ = **Py**thon Productivity for Zy**nq**
- provides a Python interface which loads and processes bitstreams to create an API type reference to firmware objects, referred to as “overlays”
- Very nice way to utilize the AXI DMA setup as PYNQ already has DMA libraries
- Resulting code needed is extremely simple!



# Conclusions and Future Work

- The Zynq MPSoC is an extremely important component of gFEX!
- Yocto OS allows for many tasks to be done more easily and efficiently in software
- Used in all types of monitoring and control of the board
- PYNQ provides a large number of libraries to help interact with the firmware from the PS
- Zynq could still do more - still thinking and trying different things!

# Backup

# PYNQ DMA Code

In [1]:
```
import numpy as np
from pynq import allocate
from pynq import Overlay

overlay = Overlay('/home/xilinx/jupyter_notebooks/Loopback/AXI_DMA.bit')
dma = overlay.axi_dma_0
```

In [2]:
```
input_buffer = allocate(shape=(8,), dtype=np.uint32)
output_buffer = allocate(shape=(8,), dtype=np.uint32)
```

In [6]:
```
for i in range(8):
    input_buffer[i] = i
```

In [7]:
```
for i in range(8):
    print("The Array is: ", input_buffer[i])
```

In [8]:
```
dma.sendchannel.transfer(input_buffer)
```

In [9]:
```
dma.recvchannel.transfer(output_buffer)
```

In [10]:
```
status = (dma.sendchannel._mmio.read(dma.sendchannel._offset + 4) & 0X01)
while(status != 1):
    status = (dma.sendchannel._mmio.read(dma.sendchannel._offset + 4) & 0X01)

status = (dma.recvchannel._mmio.read(dma.recvchannel._offset + 4) & 0X01)

while(status != 1):
    status = (dma.recvchannel._mmio.read(dma.recvchannel._offset + 4) & 0X01)

print("DMA transfer success..\n");
```

```
DMA transfer success..
```