Pixel-Standalone Julia Documentation

1. **SiPixelRawToCluster Module**

**Module Type:** Subtype of EDProducer (produces data for the Event)

**Overview**: The SiPixelRawToCluster module *processes raw data to* *generate digi and cluster data* for events. It consumes a FEDRawDataCollection token to retrieve the FEDRawDataCollection from the Event and uses two put tokens to *store the digi and cluster data into the event*. Additionally, it has another put token to store SiPixelDigiErrorsSOA if errors are included.

**Boolean Vars:**

* Is\_run2
* Include\_errors
* Use\_quality

**Main Functionality**: The module's primary task is to produce digi and cluster data using the produce function. This function takes the SiPixelRawToCluster module, the event (iEvent), and the event setup (iSetup) as input arguments.

**Functions and Structs Included:**

function produce(self:: SiPixelRawToClusterCUDA,event::Event, iSetup::EventSetup)

1. Retrieve the SiPixel Fed Cabling Map Wrapper from the event setup.
2. Extract the cabling map and modulesToUnpack.
3. Get the SiPixelGainCalibrationForHLT from the event setup.
4. Retrieve the gpugains.
5. Obtain the FedIds from the event setup.
6. Fetch the fed\_raw\_data\_collection from the Event.
7. Initialize counters:
   * fedCounter: Counts the number of valid FEDs within the range 1200 to 1349.
   * wordCounter: Counts the number of 32-bit words.
8. Create ErrorChecker to validate headers and trailers for FedRawDataCollection payloads.
9. Loop through all FEDs in the fedIds array:
10. For each FED, read its payload:
    * The payload begins with a 64-bit (8-byte) word header and a 64-bit (8-byte) FED trailer.
    * Validate the fedtrailers checkCRC bit, and the moreheaders and moretrailers bits of the payload.
11. Skip headers and trailers, focusing on the payload.
12. Store the word and FEDs in a struct named WordFedAppender, which holds the word and FED IDs.

After processing the payloads, the module calls the

make\_clusters(self.gpu\_algo,self.is\_run2,

                        gpu\_map,

                        gpu\_modules\_to\_unpack,

                        gpu\_gains,

                        self.word\_fed\_appender,

                        self.errors,

                        word\_counter\_gpu, # number of 32 bit words

                        fed\_counter, # number of feds

                        self.use\_quality,

                        self.include\_errors,

                        false) #make clusters

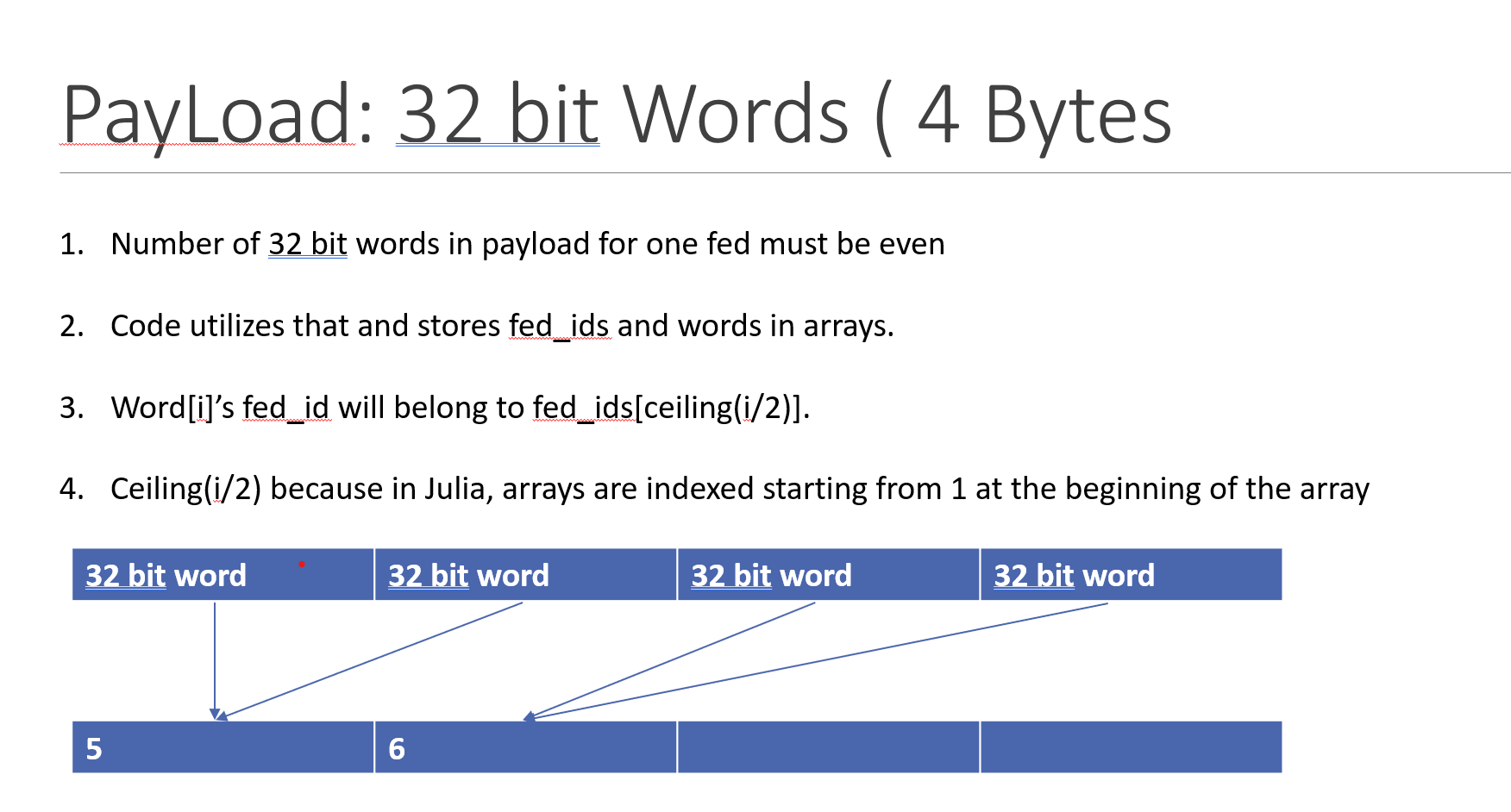
method to *create both digis and clusters* and place them within the event. If include\_errors is true, it also stores SiPixelDigiErrorsSOA within the event.

 struct WordFedAppender

        words::Vector{UInt32}

        fed\_ids::Vector{UInt8}

    end



32 bit word structure:

32 bit word for pixels not on layer 1

        [  6 bits  | 5 bits | 5 bits |  8 bits  |8 bits  ]

        [  Link    |  ROC   |  DCOL  |  PXID    |  ADC   ]

Special For Layer 1

        [  6 bits  | 5 bits | 6 bits |  7 bits  | 8 bits  ]

        [  Link    |  ROC   |  COL   |  ROW     |   ADC   ]

function raw\_to\_digi\_kernal(cabling\_map::SiPixelFedCablingMapGPU , mod\_to\_unp :: Vector{UInt8} , word\_counter::Integer,

                                word::Vector{UInt32} , fed\_ids::Vector{UInt8} , xx::Vector{Int16} , yy::Vector{Int16} ,

                                adc::Vector{Int32} , p\_digi::Vector{UInt32} , raw\_id\_arr::Vector{UInt32} , module\_id::Vector{Int16},

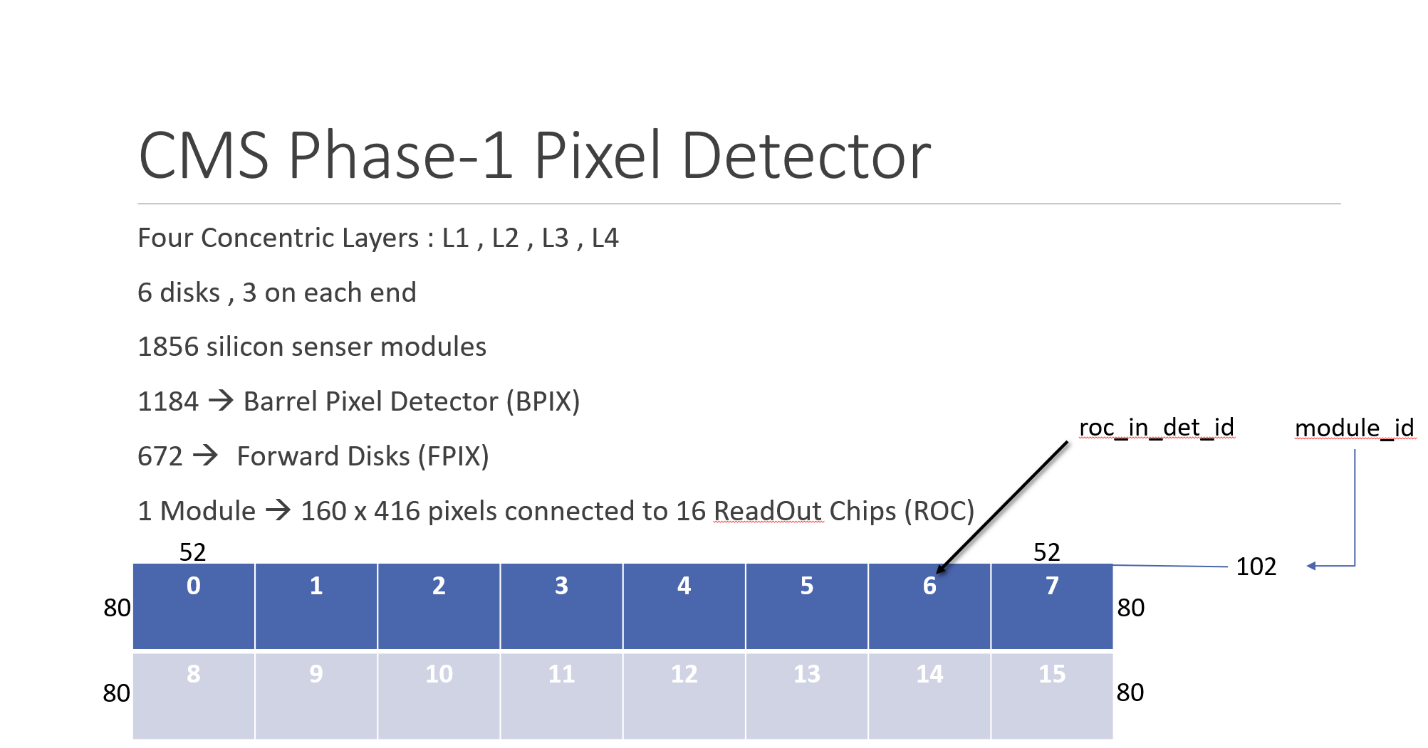
                                err::Vector{PixelErrorCompact} , use\_quality\_info::Bool , include\_errors::Bool , debug::Bool)

**Functionality**: The Raw\_to\_digi\_kernel function is responsible for *converting FED raw data to digis* by iterating over the wordfedappender and producing the SiPixelDigiSOA.

Input Arguments:

* Word Counter: The number of 32-bit words.
* Array of words: The array containing the words to process.
* Array of FedIDs: The array of FED IDs.
* Output Arrays: Arrays to store x, y, adc, packed digi, moduleId, and rawIdArr values for each word.

**Function Process:**

1. Loop Through Words: Iterate over the array of words provided by the wordfedappender structure.
2. Initialize Variables: Set x, y, adc, pdigi, rawIdArr, and moduleId to default values.
3. Extract Word Information: For each word at index i, extract the link and ROC.
4. Index into Cabling Map: Use the link, ROC, and FED ID to access the cabling map and retrieve the detector ID, which contains details about the module (160 x 416 grid pixels).
5. Determine Pixel Type: Use the detector ID to get the raw ID and determine if the pixel is in the barrel region.
6. Access ROC Information: Retrieve the ROCIdInDetUnit, the index of the ROC within the 2x8 ROC array.
7. Set Default Variables: Initialize layer, side, panel, and module variables.
   * Barrel Pixels: Extract layer and module from the raw ID. Set the side based on the module value (< 5 is -1, otherwise 1).
   * Non-Barrel Pixels: Set layer to 0 (indicating a disk), extract the panel from the raw ID, and set the side based on the panel value.
8. Determine Coordinates:
   * For layer 1, directly extract x and y coordinates from the 32-bit word.
   * For other layers, infer local pixel coordinates within the ROC from the dcol and pixel ID.
9. Frame Conversion: Use the frameconversion function to convert local pixel coordinates to global coordinates. Set the x, y, adc, pdigi, moduleId, and rawIdArr for the 32-bit word at index i.

**Figure 1:Individual Module Model**

1. **RawToClusterKernel Module**

**Functionality Overview**: The RawToClusterKernel processes inputs such as the cabling map, wordFedAppender, wordCounter (number of 32-bit words), and fedCounter (number of FEDs). It creates the SiPixelDigiSOA and SiPixelClusterSOA for further processing.

**Process Flow:**

1. Create Data Structures:
   * SiPixelDigiSOA is populated by the raw\_to\_digi\_kernal (Unpacker).
   * SiPixelClusterSOA is created.
2. Pixel Calibration:
   * Use the calibDigis function to pass the x, y, and adc arrays with SiPixelGainforHLT to get pedestal and gain values for each pixel, as these values depend on each pixel.
   * Calculate vcal using the formula:
   * vcal = adc[i] \* gain - pedestal \* gain;
   * adc[i] = max(100, int(vcal \* conversionFactor + offset));
   * conversion\_factor = (is\_run\_2) ? (id[i] < 96 ? v\_calto\_electron\_gain\_L1 : v\_calto\_electron\_gain) : 1.0f0;
   * offset = (is\_run\_2) ? (id[i] < 96 ? v\_calto\_electron\_offset\_L1 : v\_calto\_electron\_offset) : 0.0f0;
   * The conversionFactor and offset depend on the boolean variable is\_run\_2:
3. Assign Cluster IDs:
   * Each digi is assigned a cluster ID from 1 to the number of 32-bit words (number of digis) within DigiSOA.
   * The moduleStart array in SiPixelClusterSOA is populated by calling the countModules function with moduleIds for each 32-bit word in SiPixelDigiSOA.
   * The first value of moduleStart array in ClusterSOA is set to the number of modules. Each subsequent slot holds an index to the 32-bit word array containing the index of the first digi (pixel) within module j, where j is the index in the moduleStart array.
   * The number of modules and digis in DigiSOA are set using the setNModulesDigi function, which accesses the first entry in the moduleStart array and the wordCounter.
4. Cluster Formation:
   * The findCluster function groups digis into clusters within each module. It takes the following as input:
     1. Module indices, x coordinates, y coordinates, and cluster IDs from DigiSOA.
     2. moduleStart, clusterInModule, and moduleId arrays from Cluster SOA.
     3. The number of digis (WordCounter).
   * Clustering Process:
     1. A histogram container is used to bin y coordinates of pixels, with the number of bins equal to the number of columns in a module (8\*52 = 416, where 52 is the number of columns in one ROC).
     2. A cluster is defined as a group of pixels where for each pixel |difference in x| ≤ 1 and |difference in y| ≤ 1.
     3. The function loops over all pixels in the module, counting neighboring digis and storing their indices in a 2-dimensional array.
     4. It then loops over all pixels again, setting cluster IDs to the minimum value among neighboring pixels. If a change is detected, the loop is repeated. Changes are propagated, changes to neighboring pixels (e.g., if pixel a points to b, and b points to c, then a points to c).
     5. The number of clusters is determined by counting the number of pixels where clusterId[i] = i.
   * All digis are assigned a cluster ID from 0 to numberOfClusters-1, indicating the cluster they belong to.
   * The moduleId array in ClusterSOA is appropriately filled.
   * The nClustersInModule array in ClusterSOA is appropriately filled.

A screenshot of a cellphone

Description automatically generated

**Figure 2: Cluster Structure Example**

1. **SiPixelRecHitCUDA Module**

The flow of the function calls to produce RecHits is as follows:

|  |  |  |
| --- | --- | --- |
| File Name | Function | Remarks |
| SiPixelRecHitCUDA.jl | emplace(iEvent, self.tokenHit, makeHits(digis, clusters, bs, getCPUProduct(fcpe))) | 1- makeHits(digis,clusters,bs, ParamsonGPU)  2-getCPUProduct(fcpe) |
| PixelCPEFast.jl | function getCPUProduct(pixelCPE::PixelCPEFast)::ParamsOnGPU | Just returns ParamsOnGPU (includes commonparams, layer geo, average geo) |
| PixelRecHits.jl | function makeHits(digis\_d::SiPixelDigisSoA,                    clusters\_d::SiPixelClustersSoA,                    bs\_d::BeamSpotPOD,                    cpeParams::ParamsOnGPU) | Calls: 1-getHits()  2-setHitsLayerStart() |
| gpuPixelRecHits.jl | function getHits(cpeParams::ParamsOnGPU,                   bs::BeamSpotPOD,                   pdigis::CUDADataFormatsSiPixelDigiInterfaceSiPixelDigisSoA.DeviceConstView,                   numElements::Integer,                   pclusters::CUDADataFormatsSiPixelClusterInterfaceSiPixelClustersSoA.DeviceConstView,                   phits::TrackingRecHit2DSOAView) | Will explain in more details below |
| PixelRecHits.jl | function setHitsLayerStart(hitsModuleStart::Vector{UInt32}, cpeParams::ParamsOnGPU, hitsLayerStart::Vector{UInt32}) | Returns an array of hits layer start that indicated for each layer, the module that begins with it |
|  |  |  |
|  |  |  |

function getHits(cpeParams::ParamsOnGPU,

                 bs::BeamSpotPOD,

                 pdigis::CUDADataFormatsSiPixelDigiInterfaceSiPixelDigisSoA.DeviceConstView,

                 numElements::Integer,

                 pclusters::CUDADataFormatsSiPixelClusterInterfaceSiPixelClustersSoA.DeviceConstView,

                 phits::TrackingRecHit2DSOAView)

**Functionality**: Processes pixel hit data from clusters and digis, adjusts for the beam spot position, and calculates hit positions and errors. This function operates in iterations over the modules and clusters.

**Process**:

1- Fix the hits average geometery based on the average geometery of the cpeParams(detector) with respect to the beam spot coordinates.

2- Iterate over the Clusters, and set the ClusParams to default values.

3- for digis in cluster, get the x and y coordinates, update the clus params accordingly. Also, Update the charge of the cluster.

4- Call the position\_corr and errorFromDB functions

5-Set the charge of the hits and the detector index, along with local x and local y, cluster size x,y and xerr and yerr.

6- Get the Detector Parameters, update the x,y, and z global coordinates. Set the r\_global and i\_phi

|  |  |  |
| --- | --- | --- |
| File Name | Function | Remarks |
| pixelCPEforGPU.jl | function position\_corr(comParams::CommonParams, detParams::DetParams, cp::ClusParamsT{N}, ic::UInt32) where {N} | Computes the position of a cluster in the detector and applies corrections. (uses functions like local\_x, local\_y, is\_big\_pix\_x, computeAnglesFromDet(),correction(),.. |
| Phase1PixelTopology.jl | @inline function local\_x(px::UInt16)::UInt16 | convert a local x-coordinate to a global x-coordinate based on shifting (so if the pixel was on an edge, u increment the shift, then increment the shift to the new position) |
| Phase1PixelTopology.jl | @inline function is\_big\_pix\_y(py) | Function to check if a pixel is a big pixel in the y-direction.(means if its on the edge of an ROC (try dividing by 52 and check the new position of the index, if its on an edge or not (based on the last\_row\_in\_ROC variable) |
| pixelCPEforGPU.jl | function computeAnglesFromDet(detParams::DetParams, x::Float32, y::Float32) | Computes the angles with respect to the detector parameters. |
| pixelCPEforGPU.jl | function correction(sizeM1, Q\_f, Q\_l, upper\_edge\_first\_pix, lower\_edge\_last\_pix,                      lorentz\_shift::Float32, theThickness::Float32, cot\_angle::Float32, pitch::Float32,                      first\_is\_big::Bool, last\_is\_big::Bool)::Float32 | Calculates the correction factor based on cluster size, charge values, and detector parameters. |
| pixelCPEforGPU.jl | function errorFromDB(comParams::CommonParams, detParams::DetParams, cp::ClusParamsT{N}, ic::UInt32) where {N} | Computes error estimates based on cluster size and database parameters. |