ATLAS Level-1 Calorimeter Trigger
Online Software Internal Note 016
22 July 2004

Online simulation package for the Jet/Energy Module - `jemSim`

Jürgen Thomas
Particle Physics Group
School of Physics and Astronomy
The University of Birmingham
`jpt@hep.ph.bham.ac.uk`

Abstract

The package `jemSim` simulates the behaviour of the Jet/Energy Module of the ATLAS Level-1 Calorimeter Trigger within the environment of the Online Simulation [3]. It inherits from the Class `L1CaloSimModule`.

Contents

1 Introduction

The package `jemSim` simulates the behaviour of the Jet/Energy Module (JEM) [2] of the ATLAS Level-1 Calorimeter Trigger [1] within the environment of the Online Simulation. The package inherits from the Online Simulation (package `l1calo/simulation`), Class `L1CaloSimModule`. For details of the simulation package, incl. the concepts of `Process Containers` and `Process Elements` and connections using `Input Valid` methods , see [3].
As a simple example, in order to setup a data connection between between the to objects `DeserialiserBlock` and `InputProcessorBlock`, first the two objects are created:

```
   m_DeserialiserBlock = new JemDeserialiserBlock();
registerInternal("DeserialiserBlock", m_DeserialiserBlock);
m_InputProcessorBlock = new JemInputProcessorBlock();
registerInternal("InputProcessorBlock", m_InputProcessorBlock);
```

and the connection is made using a data name, here `EMTowers`:

```
   tmp = m_DeserialiserBlock->getOutput( "EMTower" );
m_InputProcessorBlock->connectInput( "EMTower", tmp );
```

This name is then checked in the `InputValid` method of the object, and data is appropriately interfaced into the object's `process` method, which then performs the actual algorithmic processing. The package is highly hierarchical.

2 Top-level class: `JemModel`

A schematic of the package showing the hierarchy of the objects is show in Fig. 1. The top-level object of the package is `JemModel`, which receives simulation real-time data via the `LVDS` and `Fan In/Out` inputs. The package receives and processes database settings for the 8 Jet Definitions (Thresholds in GeV, and Size W2, W4 or W6) and mask settings. These settings are defined in `dbL1Calo/db/DbJem.cxx`, which is used both by the simulation and by the online controlling module services `jemServices`.

Within `jemModel`, the flow of data follows the general theme of the simulation framework. Information on the required results is passed to the simulation, which then propagates upstream to process the various objects in the model in the arranged order. In general, a concept close to the JEM hardware is followed. The Jet and EnergySum algorithms themselves are includes in C-code in `JemAlgorithm`, which is interfaces to the rest of the package via `JetEnergyProcessor`.

The naming scheme still follows the JEM 0 specification, as one InputFPGA (InputProcessor) is working on data of one ϕ bin. All details of the algorithms however are valid also for JEM 1, apart from the readout which is to be updated to the new 9U ROD format. In general, many processing functions are hierarchically broken down into 'Blocks'

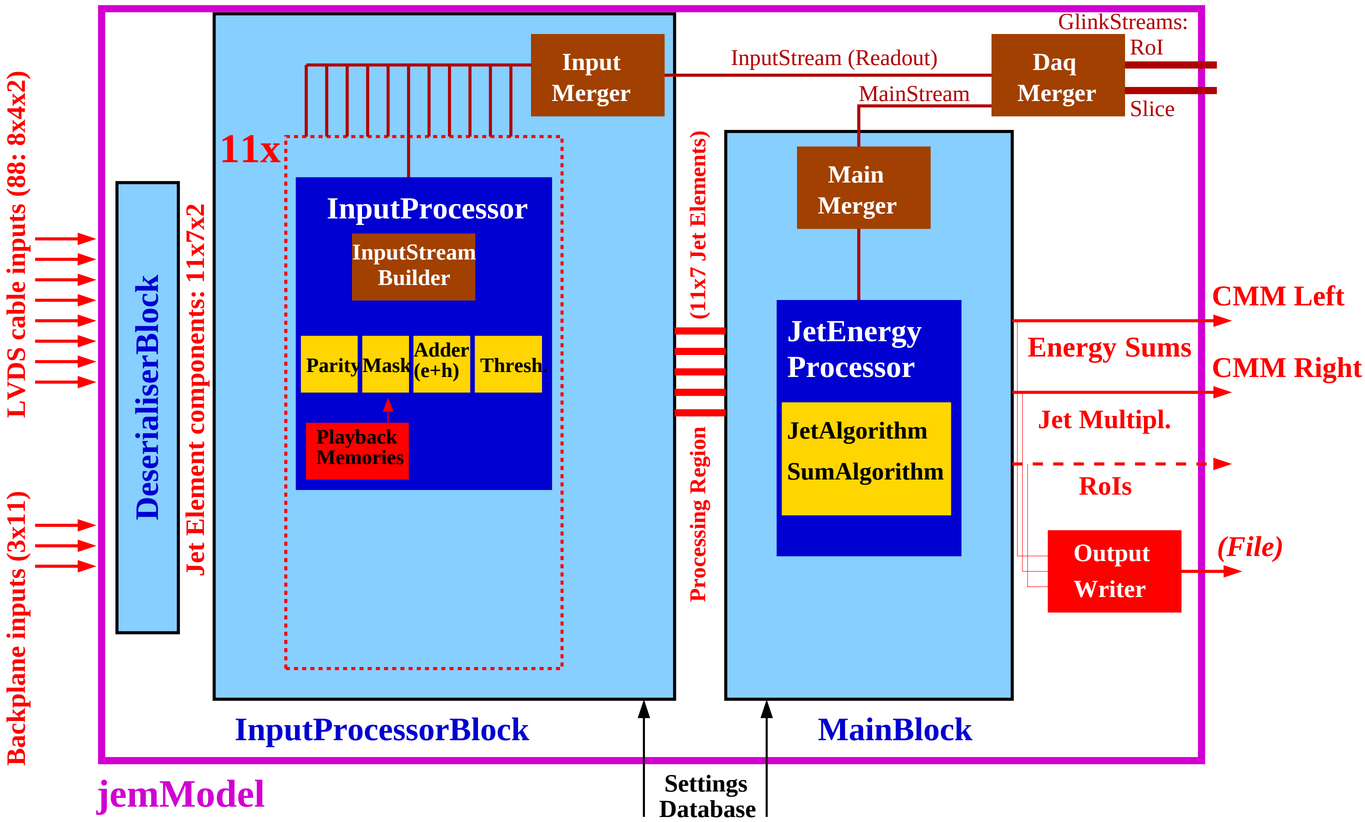

Figure 1: Package `jemSim`: Schematic hierachical representation

which contain many instances doing the same processing. E.g. the 11 InputProcessors, each working on one ϕ bin, are grouped together into an `InputProcessorBlock`.

The order of processing is the following:

1. `DeserialiserBlock`: Contains the connections between the incoming LVDS data cables and the internal mapping
2. `InputProcessorBlock`: Contains the 11 InputProcessors
3. `MainBlock`: Contains the MainProcessor. Note that the simulation is following JEM 0 architecture, so both Jet and Energy algorithms are performed in the same entity.
4. `DaqMerger`: Interface to the highest level of the readout stream-producing infrastucture. It provides the DAQ and RoI readout streams. The DAQ readout is interfaced to the `MultiSlice` infrastucture of the `Simulation` package.
5. `OutputWriter`: When the JEM module is connected to the DataGenRecipe `JemSpyOutput`, the simulation provides an output file containing all algorithm results in readable format

Internal connections are set-up between the respective blocks in the same way as in the hardware, to provide data inputs and forward data outputs to the external world. For the CMMs, the outputs are:

- `CmmLeft` : The 25 bit energy real-time-path results (called 'MergedEnergies') are connected to the backplane simulation port that forwards data to the left-hand CMM in the crate (as in hardware).
- `CmmRight` : The 25 bit jet results real-time-path (called 'MergedJets') are connected to the backplane simulation port that forwards data to the right-hand CMM in the crate (as in hardware).

Note that the test option of fan-out of energy data to both side CMMs is not implemented. If such an option is wanted, the connections need to be changed in `JemModel` (commented lines).

3 `DeserialiserBlock`

Contains the connections between the incoming LVDS data cables and the internal mapping. The connections of the input channels are made into the 7x11 η/ϕ array which later-on provides the input to the JEM algorithms. For the core region, this is done by looking at the input channels labelling (`AB, CD, ..., V`), which is forwarded in the setup from the cable labels in the database. Note that the correct connection of the half-populated cable labelled `'V'` still needs to be confirmed. The FIO data is connected via the backplane simulation providing 'EtaSlices'.

4 InputProcessorBlock

This block houses 11 `InputProcessor` and 11 `InputStreamBuilders`, as well as one `InputMerger`, which merges the 11 readout stream together. In the readout, the `SerialStream` is used, which is a boolean port of length 89. This follows the CPM simulation. Note that this format will need to be changed for the 9U ROD implementation.

4.1 InputProcessor

The algorithmic processing steps are broken down into tiny objects doing a very simple job.

1. `JemParityCheck` : Checks incoming 9-bit data for parity errors by calculating odd parity and checking with 10th bit. Sets parity error bit if mismatching.
2. `JemMask`: Apart from applying the mask settings, this object also switches between the Playback Memory and the real-time data. The mode is selected by setting `setPlaybackMode`.
3. `JemAdder` : Adds electromagnetic and hadronic data.
4. `JemThresholdCheck` : Checks against global thresholds.

The playback memories for each `InputProcessor`, meaning the eight data values for every phi bin, are interfaced with the processing chain in the function `fillPlaybackMemories`. The 'playback mode' setting is forwarded to `Mask`, which then does the switch between the data streams. The ordering of the phi bins w.r.t the Playback Memories is done in the `InputProcessorBlock`. There, potential problems with the phi direction can be addressed.

4.2 InputStreamBuilder

Each of the 11 `InputStreamBuilders` formats the incoming LVDS data, Link Error and Parity Error bits and puts them into the G-Link format for the one corresponding `InputProcessor`. The channels are filled following the order `1e 1h 2e 2h 3e 3h 4e 4h`. Note that this is not the order in the received data itself (which uses the order `1e 2e 3e 4e 1h 2h 3h 4h`), therefore a reshuffeling is performed. To follow an agreed convention, when Playback Mode is selected, by definition there are no Link or Parity Errors bits set. When a Link Error occurs, also the Parity Error bit is always set.

5 MainBlock

This is only a container for one `JetEnergyProcessor` and one `MainMerger`. This was done to match the CPM simulation (which has 8 'MainProcessors' in the equivalent Block), so the readout infrastucture could be copied.

5.1 JetEnergyProcessor

The `JetEnergyProcessor` only interfaces the algorithm itself into the OO environment of the package.

5.1.1 Algorithm

The algorithm itself is included in C-code in `JemAlgorithm`, which is interfaced to the rest of the package via `JetEnergyProcessor`. The algorithm processes the 7x11 array of input data and provides the set of results of real-time energy sums (Sum-E_X/Sum-E_Y, Sum-E_T) and jet multiplicites; and the RoI pattern, which goes only to the RoI readout. These are saved in ROIArray and passed as 'Results'.

- The Sum-E_X/Sum-E_Y algorithm uses a predefined set of values for the LUT multiplication with cos/sin ϕ. This has to match the values used in the hardware. The algorithm matches the current (July 2004) version of the JEM 1 algorithm. The implementation of the Energy Summation algorithm is within the routine `SumEnergy`.

- The Jet Algorithm produces arrays of different size due to the different jet definitions (size and thresholds) processed in parallel. These array summation are performed in the routine `Generate2x2`, `Generate3x3` and `Generate4x4`, for the window sizes W4, W6 and W8. These arrays are then processed in `FindLocalMaximum`. The RoI candidates are identified in `Set8Roi2x2` and compared to thresholds in `SetThresholdsPassed`, then finally the results are saved in `SaveJetMult`, and results are formatted and parities added, also from the Energy Summation algorithm in `FormatResults`. For debugging purposes, the incoming data array can be dumped when uncommenting the final part of `Generate2x2`, and the algorithmic results by uncommenting the middle part of `FormatResults`.

Note that the internal order of the energy sums is: SumET is E[0], SumEY is E[1], SumEX is E[2]; while the output order in all the rest of the package is: SumEX is E[0], SumEY is E[1], SumET is E[2], which is also the order on the backplane. This was kept to have the original algorithm code.

5.2 MainMerger

The `MainMerger` receives the results for the Energies, Jet Multiplicites and RoIs from the `JetEnergyProcessor`. The RoI results are connected through a port of eight. The only operation done is to add the parity bit for the real-time results on the 25th bit. The RoI values already come with the parity on each of the eight words.

6 DaqMerger

The readout streams are in general filled within the objects that have the necessary real-time data available, and then merged together to form the readout format. The readout format used is the one of JEM 0 (also JEM 1 in 2004 Combined Test Beam configuration). Each `InputProcessor` produces data for one bit of the G-Link stream. These streams are formed from the data in the `InputStreamBuilder`. The other bits come from the `MainProcessor.` and are filled in DaqMerger. As the readout G-Link packet is 89 bits long, but the filling routine uses an `int` variable to fill bit by bit, which is limited to 32 bits, the higher bits had to be put into seperate stream-filling and merging loops. The DaqMerger also includes a reshuffling of incoming RoI data to match the RoI output order of the RoI format.

7 OutputWriter

When connected to an output file via a DataGenRecipe, the package can output three types of files: First, a readable printout of the results of the algorithms, the three energy-sum values and the eight jet multiplicities, and also the eight RoI results are printed of every event when the setting `JemSpyOutput` is used for the Data Type connected to the JEM in the database setup. For example, a result for a random pattern can be:

```
   # Filename:  jep0-jem0.sim
# Generated:  18:50:52 Mon 12 Jul 2004
# By Class:  JemOutputWriter
# # Output from JEM Simulation
# Readable format (formatFull):
# SumEX SumEY SumET 8x JetMult.  8x RoIs
1 2c0 0d8 300 1 1 1 1 1 1 1 1 2FF 800 800 800 800 800 800 800
1 480 1f0 540 2 2 2 2 2 2 2 2 800 800 800 800 2FF 8FF 800 800
1 360 0f4 390 1 1 1 1 1 1 1 1 8FF 800 800 800 800 800 800 800
1 2a0 130 300 2 2 2 2 2 2 2 2 800 8FF 800 800 2FF 800 800 800
1 240 140 2b0 1 1 1 1 1 1 1 1 800 2FF 800 800 800 800 800 800
1 480 094 480 1 1 1 1 1 1 1 1 800 800 800 800 2FF 800 800 800
```

The energy-sum are printed in the order Sum-E_Y, Sum-E_X and Sum-E_T in hex format. The RoI results have a parity, therefore the value '800' represents an empty subregion.

The G-Link output streams can also be dumped as raw data. For this, the `GLinkWriter` from the ROD simulation package `rodSim` is used. To select the DAQ G-Link stream, the Data Type connected to the module in the database setup has to be switched to `GlinkJemData`, for the RoI stream to `GlinkJemRoIs`. [1]

[1] It is possible to dump the data send to the CMM by using `JemMergedJetEnWriter` by connecting the data to `mergeWriter` in the `JemModel` by hand. Note that only one output file can be connected to each JEM !

8 Test Pattern Generators

The simulation package includes data generators that produce data files. Their contents are filled into the corresponding hardware memories by the online module services, e.g. for usage in loop-back tests. As with all modules, when connected to a source data type `DataGenRecipe` in the database setup, the appropriate JEM simulation generator is called by a corresponding class in the `dbSim` package.

8.1 `JemDssGenerator`

This generator produces test pattern input files to be read into Data Source and Sink Modules (DSS), equipped with two LVDS Source Daughtercards, to source 16 channels, usually connected to fill two phi-bins. This is data of the datatype `JemLvdsInput`. The corresponding class for this generator in `dbSim` is `DbGenDssToJem`. The generator produces data matching the DSS memory buffer and LVDS link channel architecture, including valid parity at the lowest bit, to be fed into JEMs. An explanation on how to add new modes can be found in the source code. The following modes are currently implemented: (the up-to-date descriptions can be found in `iguiL1Calo/data/online-help/Generators.html` under 'LvdsDssToJem'):

1	Calibration data (all zero, all one)
2	Different ramps for each tower
3	Really random data
4	Special random data, full JEM input word range
5	Ramp up one channel (Range set by 'MaskValue')
6	Zero data in all channels
7	Ramp all channels (Range set by 'MaskValue')
8	Ramp two channels up to 255 (Jets in non-neighbour DSS)
9	Constant value for four channels ('MaskValue' as range)
10	Ramp one channel of only one DSS ('MaskValue' is DSS No.)
11	Two channels, ramping in 10 GeV steps ('MaskValue' is range)
12	Constant value for two channels ('MaskValue' as value)
13	Special pattern for RoI location check ('MaskValue' is location)
14,15	Special patterns for RoI readout offset finding (raw/fine)
15	Special patterns for Backplane crosstalk tests

Notes on the generator modes: All DSSs are filled with the same data (expect (9)). 'DataValue' is not used ! All settings via *MaskValue* in the IGUI. Special simple modes (10,11) for JEM-CMM/Sum testing. Range for ramps via MaskValue, using logical -and- (e.g. 0xf for upto 15, 0x3f for upto 63 etc.). The special RoI patterns should 'switch on' RoIs in subregions one by one by passing some jet thresholds (standard trigger menu settings).

8.2 JemInputGenerator

This generator produces test patterns to be read into the Playback Memory infrastructure of both the simulation and the hardware. It is set-up to produce data for all the 88 input channels, but most generator modes fill only the 64 core region input channels. The corresponding class for this generator in `dbSim` is `DbGenJemInput`. An explanation on how to add new modes can be found in the source code. The following modes are currently implemented: (the up-to-date descriptions can be found in `iguiL1Calo/data/online-help/Generators.html` under 'LvdsJemPlayback'):

0	Ascending bit pattern, only hadr. core ch.
1	Ascending bit patterns, all channels
2	Really random data
3	Special random data for JEM, only hadr. core ch.
4	Special random data, all channels
5	Sparse Random data
6	Ascending bit pattern, one channel only
7	Constant value for all inputs
8	Zero data everywhere
9	Two nice jets (gaussian shape), maximum ramped, had: 2x em
10	Two nice jets (gaussian shape), random maximum, range limited
11	Ascending bit pattern for ch. 30 only, range limited
12	Read physics test vector file (tower-data)
13	Four simple jets (single jet elements), ramped up to 255

Notes on the generator modes: The setting of *Data Value* from the IGUI is used as a parameter in random pattern (3)(4), Used as channel no. in (6), Used as constant value in (7); Used as range via mask and logical -and- (e.g. 0xf for upto 15) for counter in (11), for random maximum in (10)
As for the *Test Vector File* read-in (12): The current format "JEMInputMain.txt" (JEM core region only, 64 values) is used (see example file in `dbSim/data/L1CaloSlice/JEMInputMain_ttbar_1.txt`).

8.3 Stand-alone test using `dbSim`

An easy way to test the JEM simulation package without the Online Software is to run it within `dbSim`. A database setup can be found in `dbSim/data`, which includes a Jet/Energy Processor with 4 JEMs and 4 CMMs: `testdbsim_4jem_4cmm_hw.data.xml`. In order to use it, it needs to be copied onto `testdbsim_hw.data.xml`. The setup sources all JEMs with playback memory data, therefore it needs to be run from the `dbSim/data` directory with the file including appropriate generator settings `genpars_jeminput.txt`:

```
../i686-rh73-gcc32-dbg/testdbsim -d testdbsim.data.xml -g
genpars_jeminput.txt
```

All generated and simulated files can then be found in the subdirectory `L1CaloSlice`. The setup can easily be changed to run with LVDS data sourced by DSSs, by replacing `jem0-gen` with `ttc-dss1to6` in the list of `dataGenRecipes`. The settings for the DSS source pattern generation in this mode need to be passed from `genpars_lvdsjem.txt`:

```
../i686-rh73-gcc32-dbg/testdbsim -d testdbsim.data.xml -g
genpars_lvdsjem.txt
```

The setup includes a Readout Driver, which dumps data into a S-Link Sink on a DSS. This data can be found in `L1CaloSlice/ttc-dss8.sim`.

References

[1] The ATLAS Collaboration: *ATLAS First Level Trigger - Technical Design Report*, `ATLAS TDR 12, CERN/LHCC/98-14`, (1998)

[2] Jet/Energy Module (JEM) prototype specification:
`http://hepwww.rl.ac.uk/atlas-l1/Modules/Modules.html`

[3] Stephen Hillier for the ATLAS Level-1 Calorimeter Trigger: Overview of the Simulation Framework, Software Note 011
`http://hepwww.ph.qmul.ac.uk/l1calo/sweb/packages` → Simulation