Table of Contents

Upgrade Calo Trigger Work

Stage 2

Documentation

- L1 Trigger Upgrade menu development: https://twiki.cern.ch/twiki/bin/viewauth/CMS/L1TUpgradeMenuDevelopment
- SLHC Calo trigger tools: SLHCCaloTriggerTools

Code & Compilation

Instructions on how to setup your workspace for the SLHC simulations is very well (and kept updated often) in the following twiki. It will help you to get started in producing the Ntuples:

- https://twiki.cern.ch/twiki/bin/viewauth/CMS/L1TUpgradeMenuDevelopment#Ntu

This basically sets the following code:

- UserCode/L1TriggerDPG
- UserCode/L1TriggerUpgrade
- SLHCUpgradeSimulations
- CaloTrigger

Running

If you succesfully run code compilation, then your are ready to produce the Ntuples.

Plotters - L1extra

Basically the analisis depends on the following class L1UpgradeNtuple which is part of the **L1UpgradeNtuple** package. This class has pointers to the objects stored in the ntuple:

```
// L1 Extra Trees (Standard and ReEmulated)
L1Analysis::L1AnalysisL1ExtraDataFormat      *l1extra_;
L1Analysis::L1AnalysisL1ExtraDataFormat      *l1emuextra_;

// L1ExtraUpgrade Tree
L1Analysis::L1AnalysisL1ExtraUpgradeDataFormat      *l1upgrade_;
```

Then the easiest way to get access to those, is to derive a new class --your analisis class-- to access those objects per event. Here is the documentation for each data format:

- L1AnalysisL1ExtraDataFormat
- L1AnalysisL1ExtraUpgradeDataFormat

In principle is very easy to gain access to the quantities you need. For

example:

```
double isoEgEt = l1upgrade_->isoEGEt[k];
```

this gives you the energy of the isolated EG k object. Having said that you can have a look to my code to do a analysis over the current and upgrade ntuples. First get the code:

```
# first authenticate with CERN
```

Class **L12015Analysis** derives from **L1UpgradeNtuple** and implements a Loop over the events in the ntuple.

- Note 1: This is an example and the naming may not be very good since it refers to 2015 (as is UCT2015). However, there are no diferences.
- Note 2: Unfortunately, due to timing issues I reused some existing code for plotting (look at the UCT2015 section). This meant that I had to read the upgrade ntuple and generate a plain ROOT Tree to make my plots. This of course is far from ideal and slows downs things. Maybe with some time I can change this.
- Note 3: That is the reason there is a dependency on L1RateTree and UCTRateTree classes.

Have a look to rootlogon.C. It shows the procedures to load the necessary dataformats and the FWLite framework.

Plotters - plotterC

Originally developped for Stage 1 studies. These scripts are on a different CVS branch. To get this code do the following:

```
# first authenticate with CERN
```

- This branch includes the scripts to make the tau resolution studies. Apart from that there are no significant differences. The idea was to re-use as much as possible code that is present for Stage 1 stuedies (trees producers a la UCT2015)

Stage 1 (UCT2015)

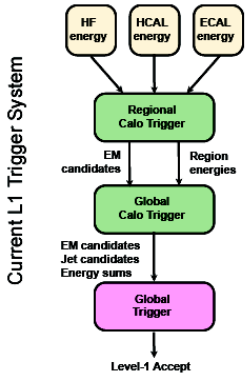
Documentation

Presentations, Twikis and code:

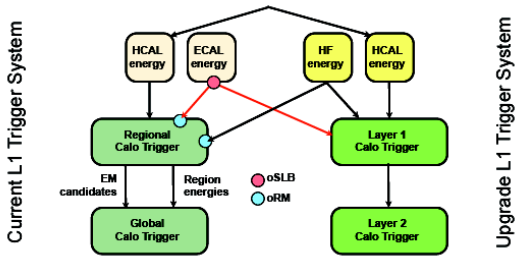
- Darin Acosta's presentation at CMS Wednesday General Meeting: WGM132: https://indico.cern.ch/conferenceDisplay.py?confId=172469
- L1 Upgrade Performance Task Force meetings (L1 Trigger Upgrade): https://indico.cern.ch/categoryDisplay.py?categId=2404
- L1 Upgrade Performance Task Force meeting (Nov 02 2012): https://indico.cern.ch/conferenceDisplay.py?confId=214955
- US CMS Trigger Upgrades - Friday, 30 November 2012: https://indico.cern.ch/conferenceDisplay.py?confId=218186
- UCT2015 TWiki: https://twiki.cern.ch/twiki/bin/viewauth/CMS/UCT2015

- CVS UCT2015 (UserCode/dasu):
  http://cmssw.cvs.cern.ch/cgi-bin/cmssw.cgi/UserCode/dasu/L1Trigger/
- Skims information:
  https://twiki.cern.ch/twiki/bin/viewauth/CMS/DatasetDefinitionTeamSkims

- Current L1 Calo Triiger:



- Proposed L1 Calo Trigger:



Code & Compilation

From the UCT2015 TWiki web page:

- For this work, I used **CMSSW_5_3_5** (@LPC analysis cluster):

```
# first set your environment for CMSSW
<cmslpc08 >
 <cmslpc08 >
 <cmslpc08 >
 <cmslpc08 >
 <cmslpc08 >   #... Here you authenticate to CERN
 <cmslpc08 >
 <cmslpc08 >
 <cmslpc08 >
 <cmslpc08 >
 <cmslpc08 >
#... Now, compile everything
 <cmslpc08 >
```

- (Authentication to CERN from LPC is needed before cheking out the code)

- Compilation under 5_3_5 worked well.

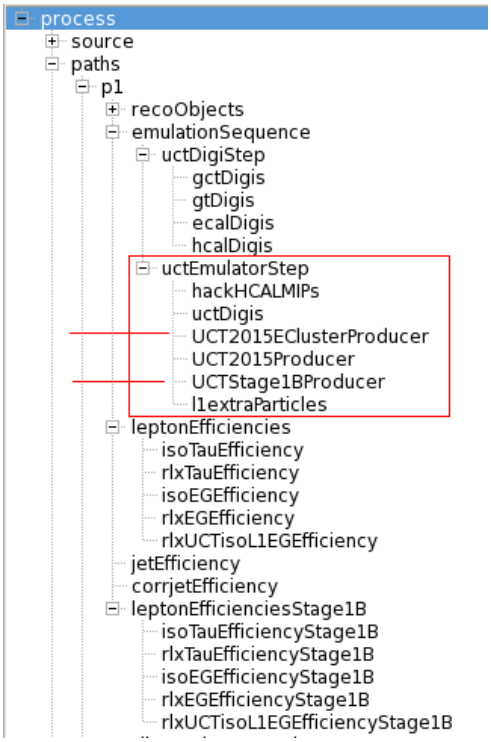
Stage 1B

The UCT guys have added a new producer, to emulate what it is known as the great **Stage1B**. To checkout the latest, have a look to the official UCT2015 Twiki page and look at the Installation instructions. It is recomended that you start from a frech release area (I personally tested in 5_3_5 and worked with the latest tag):

- https://twiki.cern.ch/twiki/bin/viewauth/CMS/UCT2015#Installation
- UPDATED You can follow the following instructions:

```
cmsenv
kserver_init
cvs co -r HEAD -d L1Trigger/UCT2015 UserCode/dasu/L1Trigger/UCT2015
cvs co -r UCT2015_v4 L1Trigger/RegionalCaloTrigger
addpkg DataFormats/L1CaloTrigger
addpkg L1TriggerConfig/L1ScalesProducers
patch -N -p0 <L1Trigger/RegionalCaloTrigger/eic9bit.patch
```

- Looking at the code, there are several cool additions. In particular, there is a new "Stage1B" producer:
  - EG/Tau Producer:
    http://cmssw.cvs.cern.ch/cgi-bin/cmssw.cgi/UserCode/dasu/L1Trigger/UC
  - E Cluster Producer:
    http://cmssw.cvs.cern.ch/cgi-bin/cmssw.cgi/UserCode/dasu/L1Trigger/UC
- This is snapshot of the new emulation sequence:



- Current Thresholds and configuration as in the default setup

Item	Stage 1	Stage 1B
Cluster size	2x1	3x3

egtSeed	5 -> EGCands	5 (egSeed) ->Clusters
regionLSB	0.5	0.5
egammaLSB	1.0	0.5 (egLSB)
HoverE	0.05 (in hardware? TDR)	0.05 (regionalHoECut)
EIC (Electron Isolation Card)	3	3
eClusterSeed (Stage1B only)	-	10
eCalLSB (Stage1B only)	-	0.5
tauLSB (Stage1B only)	-	1
tauSeed (Stage1B only)	-	5

- egSeed applies to Clusters produced by the UCT2015EClusterProducer
- egtSeed applied to newEMCands which are essentially "uctDigis".

- Isolation

Isolation Cut	Stage 1	Stage 1B
-	-	0.1 (egRelativeEMJetIsolationCut)
-	-	0.1 (egRelativeEMRgnIsolationCut)
-	-	0.1 (egRelativeJetIsolationCut)
-	-	0.1 (egRelativeRgnIsolationCut)

- Pileup

Item (default)	Stage 1	Stage 1B
puCorrect	true	true
useUICrho	true	true (hardcoded - see UCTStage1BProducer::puSubtraction)
puETMax [GeV]	7	10 (hardcoded - see UCTStage1BProducer::puSubtraction)
puETMax (Stage1B Clusters) [GeV]	-	7

- Stage 1B efficiency ntuple. Special branches:

Branch	In the C++	Definition
l1gRegionPt	'associatedRegionEt'	= C = regionEt (= region->Et() * regionLSB)
l1gJetPt	'associatedJetPt'	== C + N + S + E + W + SW + SE + NW + NE
l1g2ndRegionEt	'associatedSecondRegionEt'	max { N, S, E, W, SW, SE, NW, NE }
l1gEllIso	'ellIsolation'	-
l1gTauVeto	'tauVeto'	-
l1gMIP	'mipBit'	-
l1gRegionEtEM	'associatedRegionEtEM	= C = regionEt (region->Et()) Et already

		in physical scale
l1gJetPtEM	'associatedJetPtEM'	C + N + S + E + W + SW + SE + NW + NE
l1g2ndRegionEtEM	'associatedSecondRegionEtEM'	max { N, S, E, W, SW, SE, NW, NE }
l1gEmClusterCenterEt	'emClusterCenterEt'	centerET == eTowerETCode[at center [eta,phi]] (Stage1BClusterProducer)
l1gEmClusterEt	'emClusterEt'	no match? this seems a bug
l1gEmClusterStripEt	'emClusterStripEt'	== stripET (= centerET + S_Et + N_Et) (Stage1BClusterProducer)

Running

The UCT2015 emualtion package is a all-in-one package, so you will get the emulation done and plain ntuple generators for rates and efficiencies performance studies. The package comes with two configuration files under the test/ directory:

- makeRateTree_cfg.py: to run the emulation and produce Rate ntuples
- makeEfficiencyTree_cfg.py: to run the emulation and produce the Efficiency ntuple

These are better described in the UCT2015 TWiki page. As an example, you may run the efficiency studies configuration by doing the following:

```
cmsRun makeEfficiencyTree_cfg.py inputFiles_load=my_file_list.txt outputFile=myOutputFile.root
```

The makeEfficiencyTree_cfg.py configuration file is located under L1Trigger/UCT2015/test/

- There is no need to write in there the two arguments. However, the configuration file will use the default arguments so you better check what are those.
- If you use the default arguments, you will need to inspect and edit the cfg.py file (as show in the next section).
- Be aware that these configuration files are in constant evolution so you need to check what tag or version to use.

» I simplified a bit these configuration files and make them ready for use with Crab. This is just to get started in a more CMSSW approach:

- makeRateTreesCrab_cfg.py.txt: make Rate trees - CRAB ready (v2)
- makeEfficiencyTreeCrab_cfg.py.txt: make Efficiency trees - CRAB ready (v2)

These files include some of the data that we have available at the LPC cluster.

Data samples

The UCT2015 Producer needs **RAW-RECO** data for **Efficiency** evaluation and **RAW** for **Rate** evaluation. In this work, I used 2012 data located at FNAL: **/pnfs/cms/WAX/11/store/data/Run2012C**

- As a first test, I introduced in the cfg file, the following input data file:

```
options.inputFiles = 'dcache:/pnfs/cms/WAX/11/store/data/Run2012C/DoubleElectron/RAW-RECO/DiTau-2
```

- My default arguments for both cfg files **makeEfficiencyTree_cfg.py** and **makeRateTrees_cfg.py** were set to:

```
# Set useful defaults
options.inputFiles = 'dcache:/pnfs/cms/WAX/11/store/data/Run2012C/DoubleElectron/RAW-RECO/DiTau-2
options.outputFile = "uct_efficiency_tree.root"
options.maxEvents  = -1
```

Running with CRAB

- First thing is to make sure that you can submit to the LPC Analysis cluster via CRAB. I have set a brief page with instrucction in here.
- Second thing: those previous cfg.py will not be accepted by CRAB. I removed the argument pass option that works if using them interactively. I attached in here the slightly modified scripts.

- makeEfficiencyTreeCrab_cfg.py.txt: make Efficiency trees - CRAB ready

- makeRateTreesCrab_cfg.py.txt: make Rate trees - CRAB ready

- Data sets
  - Rates (ZeroBias3 - 2012C - RAW):
    dataset=/ZeroBias3/Run2012C-v1/RAW
  - Efficiencies (RAW-RECO)

Wisconsin Tau skim files

The Wisconsin team has setup and generated a skim dedicated for Tau studies. The files are located at their computing Tier-2. The logical path and name to those files are in the attached document. To run on those you need to add at the beginning of your file the following prefix "root://cmsxrootd.hep.wisc.edu/". So for instance, an input file would look like this:

```
options.inputFiles = 'root://cmsxrootd.hep.wisc.edu/store/user/swanson/MuTauSkim/skim_100_0_Uha.r
```

- notice the double "//". That is very important.

Ganga users: you can use the attached script and work in the same way as you do with files located at the LPC. Example:

```
ganga -i submitAnalysisCondorXROOT.py -f mutau_skim_WIs_guys.txt -c makeEfficiencyTreeCluster_vHE
```

- mutau_skim_Wis_guys.txt: Wisconsin Tau skim files

- submitAnalysisCondorXROOT.py.txt: Ganga script to subimit to Condor and use the XROOT protocol to access files

Output - Trees content

- makeEfficiencies produces a Trees that has the content described in here (UCT2015 Twiki)
- makeRate produces Trees better described in here (UCT2015 Twiki)

Plotting Tools

- The UCT2015 code comes with a series of scripts implemented in python. These scripts run fine, but be aware that they could be out-of-date as they are rapidly evolving.
- I have created some ROOT/C++ code to make the same plots. This code can be obtained from UserCode/aosorio/UCT2015
- In this repository, you can also get the CRAB configuration files to run over data the rate and efficiency producers.

Plotting Tools - CVS Branches

- Available CVS Branches and Tags of the plotting tools

Tag	Branch	Description	Latest Tag
tdr-V09	yes	Scripts used for Stage 1 performance studies - Plots as in the TDR	tdr-V11
slhc-V01	yes	Scripts used for SLHC performance studies (Stage 1 vs Stage 2)	slhc-V01
stage1B-V02	yes	Scripts used for Stage 1B performance studies (Stage 1 vs Stage 1B)	stage1B-V02

Getting the TDR scripts

* This is the way you would get the latest TDR scripts ---use to make the plots we contributed to the Trigger Upgrade TDR:

```
# first authenticate with CERN
<cmslpc08 > kserver_init
 <cmslpc08 > cvs co -r tdr-V11 -d plottersC UserCode/aosorio/UCT2015/plottersC
```

- Note: Of course you would need the data (the ntuples) that were used to fill in the histograms and graphs.

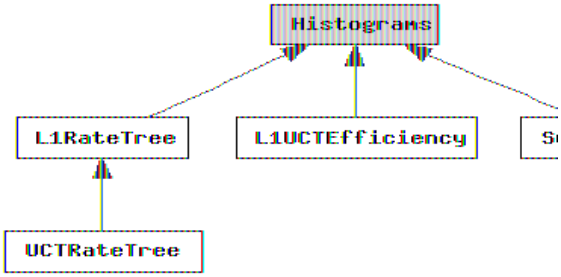
Plot Utilities Description

- An easy way to explore the code is to generate the doxygen documentation. There is a configuration file under the directory plottersC/doc. Just run "doxygen config" inside that directory.

```
<cmslpc08 > cd doc/
 <cmslpc08 > doxygen config
```

- This will generate the html documentation which you can open in your web browser by opening the file index file*plotters/C/doc/html/index.html*:

- Class inheritance overview (from doxygen)



- All this classes need to be compilided. The code has a **rootlogon.C** file that helps in compiling and loading the needed library. Edit the *rootlogon.C" script to swith on "1" or switch off "0" the compilation:

```
void rootlogon()
{

  rootlogon(0);
  //don't compile: 0 compile: 1

}
```

EvalNormalization> L1 Normalization factor (-> Hz): 126.884

- A brief description of these clases follows:

Class	Comment
Histograms	Basic histograming functions, 1D,2D histograms containers
L1RateTree	Class to perform analysis on the RateTree ntuples (L1)
UCTRateTree	Class to perform analysis on the RateTree ntuples (L1 Upgrades)
L1UCTEfficiency	Class to perform analysis on the EfficiencyTree ntuples (L1 Upgrades)
SumsEfficiency	Class to perform analysis on the SumsEfficiencyTree ntuples (Energy Sums Upgrades)

- In addition, there are two small classes **RateHisto** and **EffGraph** and will contain the histograms for rates and efficiencies respectively, and provide the ability to work within the ROOT framework (as they derive from the TObject class).

- All these classes provide a nice set of tools to make rate, efficiency plots and anything you need. There are two approaches to make plots: usign the **Draw** command from root and **Looping** over the event contents. Both ways are equivalent. One may be simpler than the other but sometimes you need access to individual events to perform any debugging.

- There are several scripts provided in this package. They are:

Script Name	Comment
runAnalysis_XX_WP.C	The runAnalysis scripts generate the rate plots for EG, TS(taus) Jets, Sums - The WP label means Working Point
plotEfficiencies_XX_WP.C	The plotEfficiencies scripts generate the efficiency plots for EG, TS(taus) Jets, Sums
plotTurnonCurve_XX_WP.C	The plotTurnonCurve scripts generate the turn on curves for EG and TS(taus)
plotResolution_TS.C	The plotResolution scripts generate the resolution plot TS(taus)

- These scripts basically contains all the necessary ingredients to generate any of these performance plots. More can be done and added. You are encourage to do it so. Please do!

- A few other scripts provide some help in keeping all the results in a well organized manner. This is the case of **prepareArea.py** which works in the following way:

```
./prepareArea.py -p feb-xx
```

- It creates under the directory **results** a subdirectory with path name **feb-xx** for example. All your results will be stored in that directory keeping things in a organized way. A tree of subdirectories is created depending on the type of study, the type of object, and type of study. Finally all plots are split into directories according to their extension.

- Although it is not required for compilation, a **Makefile** is included in this package. Type **make clean**, and this will remove all the unwanted existing binaries.

Special notes

- In order to keep consistency between the **selection** you apply to generate the rate/efficiency plots, they recomended way to proceed starts by running the Rate scripts first. These scripts write a log file which contains the selection you applied. This file is stored under the **config/** subdirectory.

- Then, you will need to run over that file a script **rate2efficiency.py**. It converts maps out the variables used in the rate tree to efficiency tree variables (not nice but it is the way this works). The obtained file will be ready to be read by the efficiency/turn-on curve scripts.

- Efficiency/Turn-on curves scripts loop over the different selections.

- This means that in principle you can generate as meany files/plots you need for your study.

- If the Ntuples are updated with NEW branches, you will need to manually edit the affected classes: **L1RateTree.h**, **UCTRateTree.h** and/or **L1UCTEfficiency** (SumsEfficiency). Otherwise you may have trouble running the scripts.

Rate normalization factor

$$f = PSF \times \frac{1}{N_{LS} \cdot \Delta t_{LS}} \times \frac{Lumi_D}{< Lumi >}$$

- where:

Item	Description	Value
PSF	Prescale total factor	2 (L1) x 23 (HLT) x 4 (Zerobias split 1,2,3,4) (*)
N_LS	Number of Lumisections (eval in script)	eval: use certified lumis. check json file
Delta t_LS	Delta time of a Lumisection	23.35 s (*)
lumi_D	Desired inst. luminosity	2.0 x 10^24 cm-2 s-1 (*): as in the TDR
< lumi >	Average inst. luminosity in run (eval in script)	eval: script calculates it

(*) known before hand

- Configuration needed to run the Rate calculation with normalization factor ON. The following example uses **plottersC/plotRates_EG_Rlx_Stg1B.C**:

```
if ( 0 ) {

   l1->SetCalibration( 1.0 );

 } else {

   float preScale = 2.0 * 23.0 * 4.0;
   l1->SetNormalizationConstants(200.0, preScale, 23.3570304);
   l1->EvalNormalization(49,112);

 }
```

- preScale = is the prescale factor
- The method **SetNormalizationConstants** fixes the a) desired luminosity (in units of 10^32 cm-2 s-1) b) prescale factor c) the time in a Lumisection
- **EvalNormalization** this method calculated the normalization factor in the valid LS range: LS_i to LS_f. In this example the valid range of lumis is [49,112]
- The previous method sets the normalization factor and multiplies the final rate histogram by this value. In this example, it does it only to the object **l1** is pointing to (in the context of this example, l1 is the pointer to Current L1 rate evaluation object).
- For the other objects (uct Stage 1, Stage 1B, etc) you will need to set the normalization factor that is already calculated:

```
float norm_factor  = l1->;
//
uct-> norm_factor ; //Both L1 & UCT use the same calibration factor
// of course the previous line has to be used  doing calling the Loop method -which produces the
```

- If you set valid lumisections, then your plot has to select events only in this window:

```
 l1->Loop("MaxIf$( pt , pt >0 )",,binning,"pt");
 [...]
 uct->Loop( Command.str().c_str(), , binning, plot_name.Data() ); // regional
```

- This assuming that you have generated your Ntuples from all available data (with CRAB you can select the valid lumi range, but not with Ganga).

- Finally: with the new scale, you may need to change the min,max of your Y-axis. You can do it in the script by setting the new range to

```
//Now Draw and compare All

uct->ComparePlots( v_rates, "EG Rates", filenamePNG );
```

- For the ZeroBias dataset run 198609 that we analyzed to evaluate rates, we got a normalization factor of (for a desired inst. luminosity of 2.0 10^34 cm-2 s-1):

```
EvalNormalization> L1 Normalization factor ( -> Hz):
```

This topic: Main > L1TriggerUpgradeUCTWork
Topic revision: r49 - 2013-05-01 - AndresOsorio