Table of Contents

How to create HLT TCKs

This page describes the standard procedure that HLT piquets should follow to produce split TCKs for 2016 datataking

Producing a dataset with at least one accepted event per HLT line

This step is not normally needed, as the existing dataset should be sufficient. Please consult with one of the HLT experts if in doubt.

There is a script in `Hlt/HltPiquetScripts` which does this. `scripts/Moore_Hlt1Hlt2_MakeDataForTCK_Parallel.py` This script runs Moore with a configurable number of parallel processes. It keeps track of which lines have fired, and only writes out an accepted event if it is the first time for any of the lines. It is slightly more efficient though, since it doesn't actually require that the line fires -- only that the next to last algorithm has been executed. First step is to make sure that we have suitable input data. The process is most efficient if we start from a mix of raw data from the different streams. E.g,.

```
nsls /castor/cern.ch/grid/lhcb/data/2015/RAW/TURCAL/LHCb/COLLISION15/164699/ | head -n 5
```

Then copy the data to `/scratch/hlt/make_tcks` on the online system. From the online system do

```
getpack Hlt/HltPiquetScripts
```

You probably want to do this in a screen session:

```
lb-run Moore v25r1 python Hlt/HltPiquetScripts/scripts/Moore_Hlt1Hlt2_MakeDataForTCK_Parallel.py
```

There will be an .mdf and .log file for each Moore process, and they will be filled in real time.

Creating and testing split HLT TCKs.

First, have a careful look at the JIRA task for the TCK creation. Take note of any special instructions mentioned there, which might override some of the following.

Before you begin, you should have an appropriate hex key for the new TCKs. You can e.g. use the latest released Moore (v25r2 in this example) with `iTCKsh` to find out:

```
lb-run Moore/latest iTCKsh
>>> listConfigurations()
.... ....
MOORE_v25r2
    Calibration_VeloMicroBias
      0x11301715 : b95ffa736bfc93d10bae16e76a3a6c96 : Prescale = 1
    Physics_pp_May2016
      0x11291600 : 8eed26d4c1e458200a7bb0ff69deeaf3 : Hlt1, Physics_pp_May2016, 0x1600
      0x21291600 : c3269de2a6f7e0bd687a82af0879778d : Hlt2, Physics_pp_May2016, 0x1600
```

Now if we want to make a new set of TCKs with the 0x1600 L0 TCK, we would use the keys 0x112a1600 and 0x212a1600. If you are making a pair of Hlt1 and Hlt2 TCKs, make sure to use the same number modulo the first digit (never mind if you skip some numbers). (For more details on the TCKsh functions, see this page.)

Setup the Moore environment

Figure out from the JIRA task which tags for Hlt and Moore to use. Normally, the Hlt tag matches the JIRA task ID and the Moore tag matches the latest release, as in the example below.

```
TASK=LBHLT-341
Hlt_tag=$TASK
Moore_tag=v26r4
```

Clone the trigger-dev project, modify the `configuration.mk` and compile.

```
git clone ssh://git@gitlab.cern.ch:7999/lhcb-HLT/trigger-dev.git $TASK
cd $TASK

# prepare configuration.mk with the right tags
sed -i -e '/^NIGHTLY_/g' -e "s/^PROJECTS *=.*/PROJECTS = Hlt:$Hlt_tag Moore:$Moore_tag/g" configu
git diff  # check this is as expected
make  # do NOT put -j
```

Restarting the TCK creation

If a problem is found with the configuration in the TCK, the procedure is to recreate the tag and start again. There is no need to repeat the cloning as done above, but it is sufficient to fetch, checkout again the tag and rebuild:

```
cd Hlt
git fetch --tags
git checkout $Hlt_tag
cd ..

make  # do NOT put -j
```

Before restarting, consider removing the logs and intermediate outputs with

```
rm -i *.{log,csv,mdf,txt,diff}
```

Setup the TCK creation options

The scripts to create and test the Hlt TCKs live in the Moore project under `Hlt/Moore/tests/options/TCK/`
You can see them e.g. with

```
Moore/run find '$MOOREROOT/tests/options/TCK'

CreateTCK1AndEntry.sh  CreateTCK1.py  CreateTCK2AndEntry.sh  CreateTCK2.py  CreateTCKEntry.py  Re
```

You will normally need to edit only `TCKOptions.py`, so copy it locally

```
# lb-run --nightly-cvmfs --nightly lhcb-head Moore/HEAD bash -c 'cp -v $MOOREROOT/tests/options/T
Moore/run bash -c 'cp -v $MOOREROOT/tests/options/TCK/TCKOptions.py .'
```

Open `TCKOptions.py` and make sure the correct options are used:

- The L0TCK is correct
- Your proposed new TCK numbers are correct
- The labels for the TCKs are meaningful and follow the template
- The HltType is meaningful and follows the template
- The ThresholdSettings, DDDBtag and CondDBtag do not normally need to be updated.

- The TCKData points to the location where the config.cdb is, see below for instructions how to get the config.cdb.

If you need to customise scripts used in the next steps, copy them locally and modify the commands as appropriate.

Prepare the input data

The starting point is a small sample of real data events, which is selected such that all HLT lines get to their next to last algorithm. It is currently at

```
/eos/lhcb/wg/HLT/2016CommissioningDatasets/OneAcceptedPerLine_v25r1_Physics_pp_May2016_190416
```

The script `RedoL0.py` uses this sample by default. If you need to change that, you should copy the script locally.

Now run

```
Moore/run gaudirun.py '$MOOREROOT/tests/options/TCK/RedoL0.py' TCKOptions.py 2>&1 | tee RedoL0.lo
```

It should produce a file RedoL0.mdf, which will be the input to the next step.

Create the TCKs

The next step is to use the output .mdf from RedoL0.py to create the HLT configuration using CreateTCK1.py.

Obtain the last `config.cdb` file from svn

```
mkdir -p TCKData
svn export svn+ssh://svn.cern.ch/reps/lhcb/DBASE/trunk/TCK/HltTCK/config.cdb TCKData/config.cdb
```

Now you should be ready to run

```
Moore/run gaudirun.py '$MOOREROOT/tests/options/TCK/CreateTCK1.py' TCKOptions.py 2>&1 | tee Creat
```

This should write the new TCK (unmapped to a number) into the `TCKData/config.cdb` file. Pay attention and report all **ERRORs** and any **WARNINGs** that you are unsure of to the corresponding JIRA task.

If there is an issue with this, or any subsequent step, **it is safest to copy again the config.cdb file, fix the problems and start again**.

Now you need to map the configuration to a TCK number. The `CreateTCKEntry.py` script finds the configuration ID from the log of the previous step and uses the TCK you defined in `TCKOptions.py`

```
Moore/run python '$MOOREROOT/tests/options/TCK/CreateTCKEntry.py' CreateTCK1.log Hlt1TCK --option
```

The next step is to test that we can actually run from this TCK.

```
Moore/run gaudirun.py '$MOOREROOT/tests/options/TCK/TestTCK1.py' TCKOptions.py 2>&1 | tee TestTCK
```

This should produce an output .mdf file called `TestTCK1.mdf`. We will use this file as input to produce the Hlt2 TCK.

Now you should be able to run the remaining steps to create/test the Hlt2 TCK in one go

```
Moore/run gaudirun.py '$MOOREROOT/tests/options/TCK/CreateTCK2.py' TCKOptions.py 2>&1 | tee Creat
Moore/run python '$MOOREROOT/tests/options/TCK/CreateTCKEntry.py' CreateTCK2.log Hlt2TCK --option
Moore/run gaudirun.py '$MOOREROOT/tests/options/TCK/TestTCK2.py' TCKOptions.py 2>&1 | tee TestTCK
```

Once you are comfortable with all of the above, you can of course wrap all the steps in one script, in case you need to redo the procedure, e.g. to pick up some package updates or other changes.

Create derived TCKs (optional)

Follow this section in case you need to create TCKs where you modify the L0 TCK and / or some properties like prescales.

In all cases here it is always recommended to create a script, e.g. `update_tck.py` and run it with

```
Moore/run python update_tck.py
```

To modify some properties and create a new TCK, follow this example:

```
from TCKUtils.utils import *
cas = ConfigAccessSvc(File='TCKData/config.cdb', Mode='ReadWrite')

updates = {
    'Hlt1BEMicroBiasVeloPreScaler': {'AcceptFraction': '0.01'},
    'Hlt1BENoBiasPreScaler': {'AcceptFraction': '0.01'},
}

old_tck = 0x113B1620
new_tck = 0x113C1620
new_label = "Hlt1, protonHelium_pPb_2016, November, BEMicroBias 0.01, LBHLT-77"
new_id = updateProperties(old_tck, updates, new_label, cas=cas)
createTCKEntries({new_tck: new_id}, cas)
```

If you need to create a new TCK with a different L0 TCK, follow this example:

```
from TCKUtils.utils import *
cas = ConfigAccessSvc(File='TCK/HltTCK/config.cdb', Mode='ReadWrite')

Updates = [
            # old TCK , new L0TCK, label
            (0x11601707, 0x1708, "Hlt1, Physics pp August 2017, 1500b, with SumET Prev. no Hersch
            (0x21601707, 0x1708, "Hlt2, Physics pp August 2017, 1500b, with SumET Prev. no Hersch
            (0x116017A7, 0x17A8, "Hlt1, Physics pp August 2017, 1500b, with SumET Prev. and Hersc
            (0x216017A7, 0x17A8, "Hlt2, Physics pp August 2017, 1500b, with SumET Prev. and Hersc
          ]

def UpdateTCKWrap(old_tck, new_L0, new_label, cas):
    new_id = updateL0TCK(old_tck, new_L0, new_label, cas)
    # keep the HLT TCK same and update the L0 one
    new_tck = (old_tck & 0xFFFF0000) | new_L0
    return new_tck, new_id

new_tcks = {}

for up in Updates:
    new_tck, new_id = UpdateTCKWrap(*up, cas=cas )
    new_tcks[new_tck] = new_id


createTCKEntries(new_tcks, cas)
```

If you need to combine both of the things above, follow the second example and add/change the following lines:

```
updates = {...}
...
new_id = updateL0TCK(old_tck, new_L0, new_label, cas, extra=updates)
```

Produce dumps and diffs

You can do this from the TCKsh interactively, but it is better to make a simple script in case you need to repeat it. Still in `MooreDev_vXrY`, create `diffs.py` with content like

```
from TCKUtils.utils import *
cas = ConfigAccessSvc(File='TCKData/config.cdb')

def dumpdiff(new, old):
    dump(new, file='dump_{:#08X}.txt'.format(new), cas=cas)
    diff(old, new, human=True, file='hdiff_{:#08X}_{:#08X}.diff'.format(old, new), cas=cas)
    diff(old, new, file='diff_{:#08X}_{:#08X}.diff'.format(old, new), cas=cas)

dumpdiff(new=0x10011600, old=0x11361609)
dumpdiff(new=0x20011600, old=0x21361609)
dump_flow(0x20011600, 0x10011600, 'flow_0x20011600.txt')
```

Modify the TCKs as appropriate and run the script

```
Moore/run python diffs.py
```

Share the diffs and dumps

- Diffs with previous TCKs should be attached to the JIRA task
- A mail should be circulated to lhcb-hlt-software and lhcb-hlt-piquet for review by the trigger WG liaisons and trigger experts
- Share and discuss the log files, and warnings, errors etc., if any, in the JIRA task

Create and test the functor cache

Work in progress, not part of the procedure yet

```
git lb-use Moore
git lb-checkout Moore/${Moore_version} Hlt/HltCache
tcks="0x10011600;0x20011600"
echo "set(tcks ${tcks})" > Hlt/HltCache/tcks.cmake
make install

Moore/run bash -c 'cp -v $MOOREROOT/tests/options/TCK/TestCache.py .'
# Edit TestCache.py as needed
# Copy over RedoL0.mdf
Moore/run gaudirun.py '$MOOREROOT/tests/options/TCK/TestTCK1.py' TestCache.py 2>&1 | tee TestCach
Moore/run gaudirun.py '$MOOREROOT/tests/options/TCK/TestTCK2.py' TestCache.py 2>&1 | tee TestCach
```

Adding the new TCKs to TCK/HltTCK/config.cdb

Finally, you will want to copy these TCKs into the config.cdb in the TCK/HltTCK package.

Checkout the head of TCK/HltTCK

```
# still from MooreDev_${Moore_version}
getpack TCK/HltTCK head
```

If you need to create a new TCK with a different L0 TCK, follow this example:

Create a file `copyTCK.py` with content following this example:

```
from TCKUtils.utils import *
source_cas = ConfigAccessSvc("ConfigAccessSvcSource", File = 'TCKData/config.cdb') ## copy the ne
target_cas = ConfigAccessSvc("ConfigAccessSvcTarget", File = 'TCK/HltTCK/config.cdb', Mode = 'Rea

# either of these would work
# you can use a glob if you have many TCKs to copy, e.g. for source in ("0x*12a1600"):
TCKs_to_copy = [0x212a1600]
for source in TCKs_to_copy:
    glob = 'TCK/0x%08x' % source if type(source) == int else "TCK/" + source
    copy(source = source_cas, target = target_cas, glob = glob)
```

Then run it like so

```
Moore/run python copyTCK.py
```

Now you should check again that TCK/HltTCK/config.cdb contains what you want.

```
svn diff TCK/HltTCK
svn diff TCK/HltTCK/config.cdb --force --diff-cmd lb-run -x "Moore latest cdb_diff"
Moore/run iTCKsh TCK/HltTCK/config.cdb
>>> listConfigurations()
```

Check that all desired new TCKs appear and there is nothing too much. (For example, you may want to copy only the Hlt2 TCK).

Add an entry in the release notes in `TCK/HltTCK/doc/release.notes`. Give a short description mentioning the JIRA task, and copy the lines for the new TCKs from the `listConfigurations()` output.

Commit your changes to TCK/HltTCK.

```
commit TCK/HltTCK -m "Add TCK(s) for this and that, JIRA-NNNN"
```

Keep the JIRA tasks, mailing lists and Moore release manager informed. Proceed to the next steps of testing at the pit before asking for a new release of TCK/HltTCK.

How to create a new

The thresholds are set by the Hlt group. The L0 group is responsible for releasing the new options files and updating the L0DU firmware. Inform the L0 piquet, OIlivier Deschamps and Regis Lefevre that a new L0 TCK is needed.

```
lb-dev Moore <latest>
cd MooreDev_<latest>
getpack TCK/L0TCK head
cd TCK/L0TCK/options
```

Look at one of the options files to get the syntax. It is probably best to start with the L0TCK which was recently in use. Copy this, increment the TCK number and make the adjustments needed. Include the new opts file in L0DUConfig.opts . Test the TCK by creating the Hlt TCKs as described above. If only the L0 part of the Hlt TCK changes, you can follow the instructions in UpdateL0TCK .

FAQ

You created an Hlt TCK but you need it with several L0 configurations

There is no need to go through the whole procedure of creating TCKs. Use the commands described in UpdateL0TCK, then run only the testing part.

Segfault when testing different L0 configurations

This often happens when going to a tighter L0 TCK. Delete the output file of RedoL0.py before running it.

This topic: LHCb > CreateSplitHltTCKs
Topic revision: r30 - 2018-04-13 - RosenMatev