Build/Deploy your CRIO-EPICs system using Nheengatu

Dawood Alnajjar
LNLS - SOL

Resources

- https://gitlab.cnpem.br/SOL/Projetos/nheengatu
- https://cnpemcamp.sharepoint.com/sites/lnls/groups/sol/SitePages/Nheengatu%20project%20-%20EPICS%20support%20for%20CRIO.aspx

Deploy steps

- Setup your CRIO with NI MAX
- Make Labview project and generate bitstream and header file
- Use template and INI auto-generation scripts to generate skeleton configuration files
- Modify template and INI files
- Generate Hierarchy structure for your CRIO on the nfs server setup-bl folder
- Move configuration files to their respective locations

Windows 10 - Labview 2019 32-bit

- Use your remote desktop client to open tesla-VM. On linux, you can launch labviewvm.sh script (https://gitlab.cnpem.br/SOL/CRIO/crio-utils.git) to open virtual machine on tesla or run the following command
  - Command executed by the script -> *rdesktop -g 1900x1040 tesla-VM*

(Make sure virtual box vm has already been started on tesla)

Setting up your CRIO – NI MAX

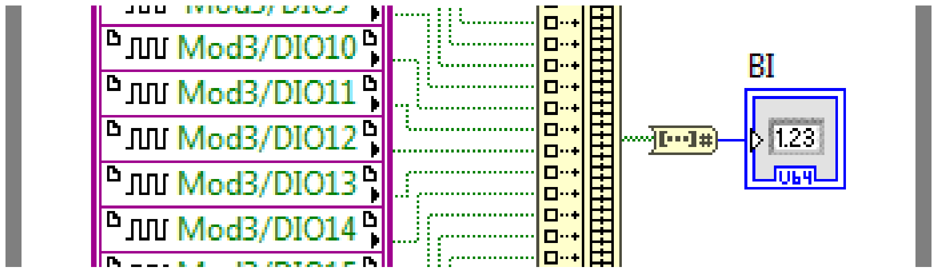
- Upon locating your CRIO in NI MAX, perform the following
  - Hostname should be BEAMLINENAME-LOCATON-CRIOALIAS (e.g. SOL-A-CRIO1)
  - Enable Secure Shell Server (sshd)
  - Update firmware to 6.5.0f0
  - Add NI CompactRIO 19.5 - June 2019 software

Create your CRIO labview project

- Reference : https://gitlab.cnpem.br/SOL/LabViewRT/crio-linux-labview
  - Open project and save so all absolute paths can be updated locally OR clone the project to the C:/ folder (project original location)
- For projects from scratch, import llb files
  - Add fpga-lib.llb to FPGA target
    - FPGA Target > Add > file > crio-linux-labview\llbs\FPGA Target\fpga-lib.llb then select fpga-lib.lvlib
  - Add rt-lib.llb to RT Compact RIO target
    - RT Compact RIO target > add > file > crio-linux-labview\llbs\RT CompactRIO Target\select rt-lib.llb then select all VIs in the list
  - Now all developed VIs should appear in the quick drop (ctrl+space)
  - Use the project in the git repository as reference

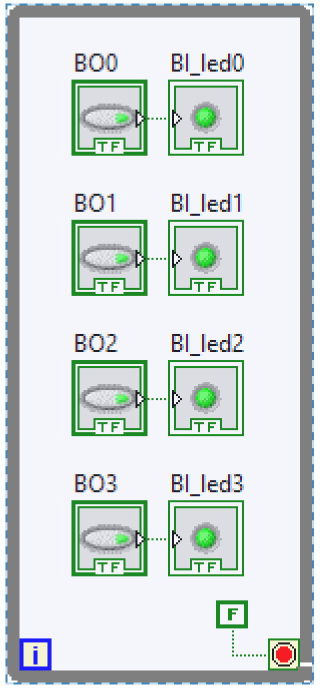
FPGA VI rules of thumb (1 / 6)

- Ref : fpga_all_example.vi, fpga_all_example2.vi
- BI **must** have a **keyword** that they will start with (i.e BI)



BIs must be concatenated into 1 single 64-bit U64
Build array followed by boolean array to number
Can be used when there are too many BIs

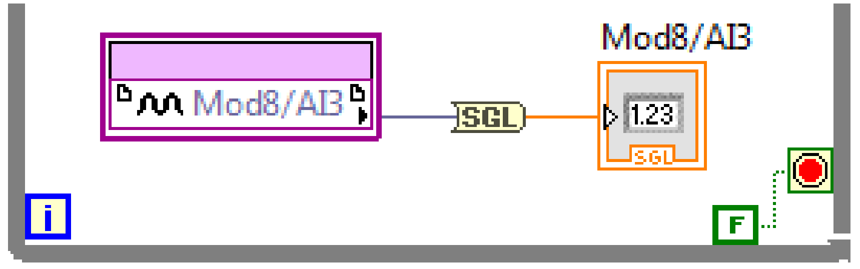
OR/AND



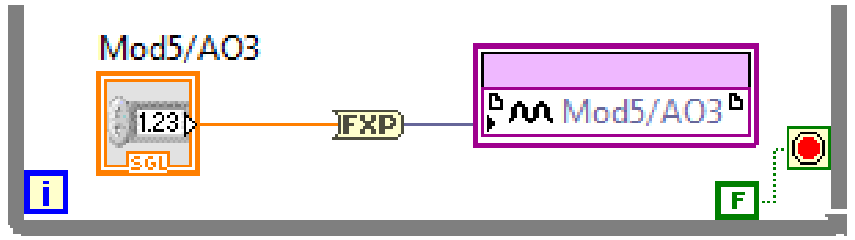
Connect straight to Boolean indicator

FPGA VI rules of thumb (2 / 6)

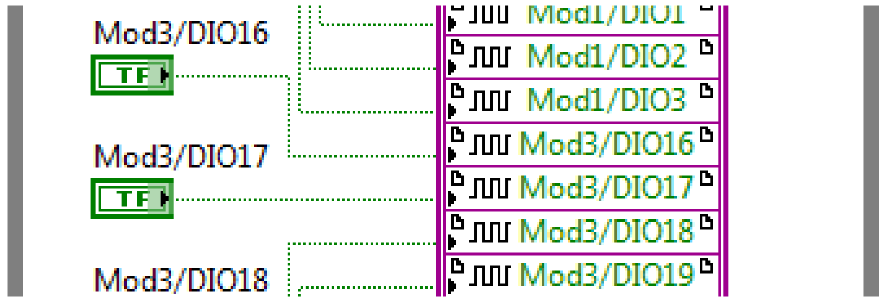
- Ref : fpga_all_example.vi, fpga_all_example2.vi
- Float AI, Float AO, BO must have a keyword that they will start with (i.e AI, AO, BO)



AI must be converted to single precision floating point



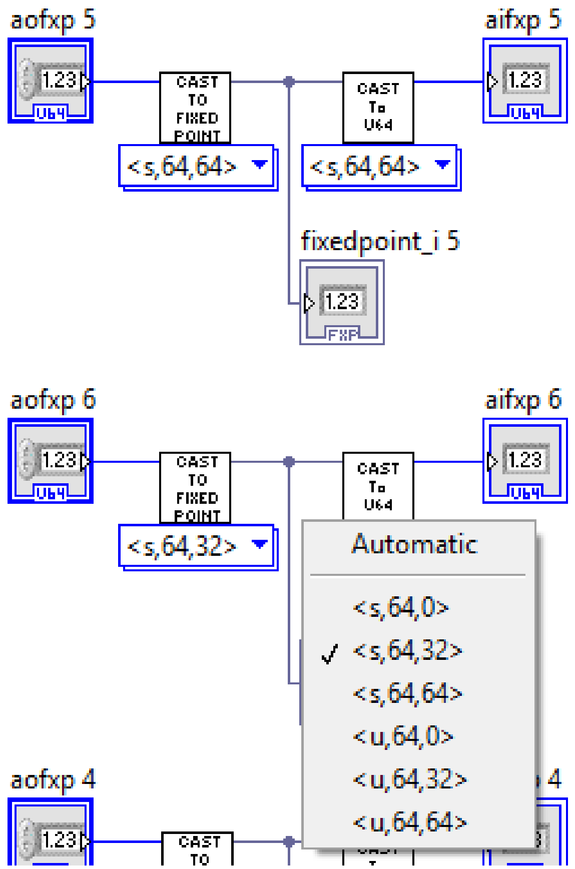
AO must be converted back to fixedpoint from single precision floating point



BOs output as is

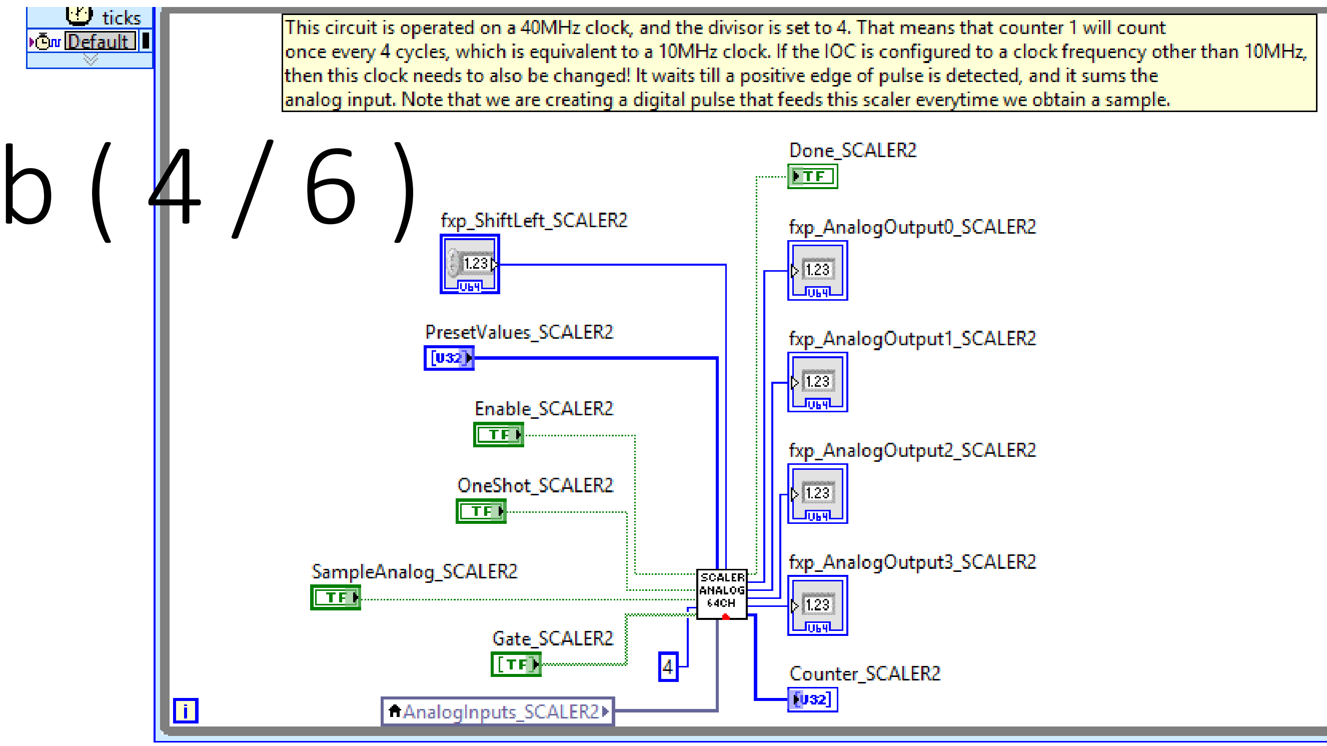
FPGA VI rules of thumb (3 / 6)

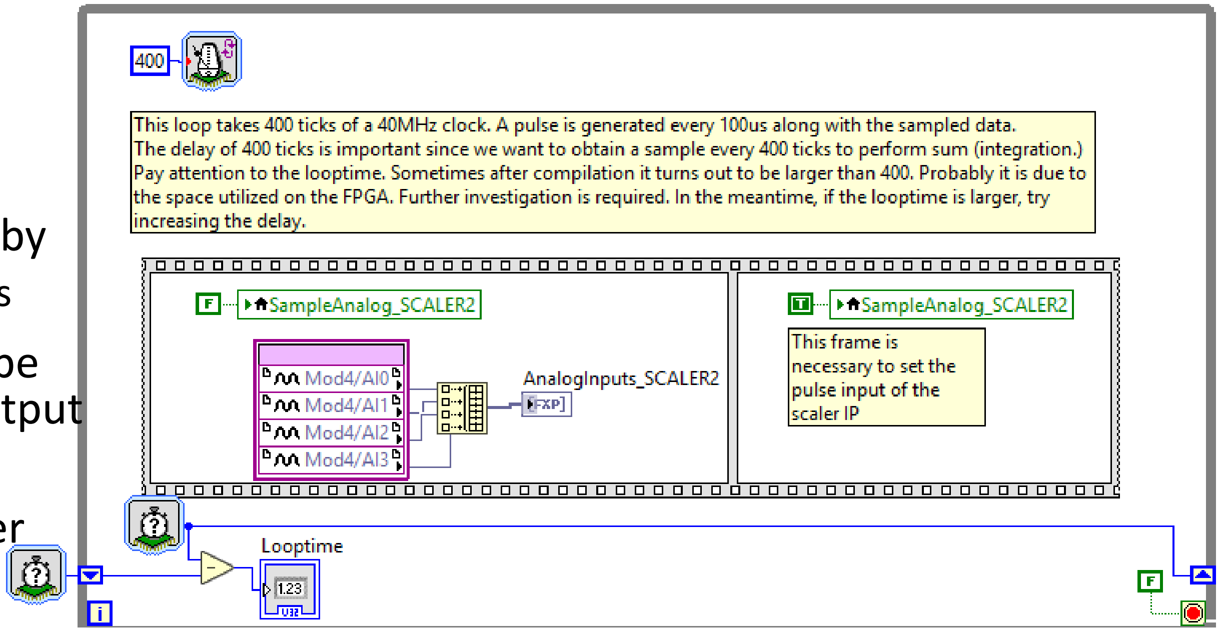
- Fixed-point AI, AO
  - Developed FPGA IP: Cast to U64, Cast to Fixed-point
  - 6 variations of each: integer length 0,32,64 signed and unsigned
- Choose most appropriate
  - FXP: supported in single cycle timed loop, higher precision, smaller range
  - SP: larger range
- Note that double is only 52-bits of precision
- Values up to 52-bits are reconstructed in Nheengatu without losing a single bit of precision!



FPGA VI rules of thumb (4 / 6)

- devloped lib VI
  - Scaler64_digital.vi
  - Scaler64_analog.vi
- Inputs/outputs
  - Enable_<SCALERKEYWORD>XXX
  - OneShot_<SCALERKEYWORD>XXX
  - Gate_<SCALERKEYWORD>XXX (64 var array)
  - Done_<SCALERKEYWORD>XXX
  - Counter_<SCALERKEYWORD>XXX
  - Preset_<SCALERKEYWORD>XXX
  - Pulse_<SCALERKEYWORD>XXX
  - Divisor
- Inputs and outputs can be automatically generated by
  - Right clock block > create > all controls and indicators
- If not all 64 counters are used, the logic can simply be optimized by reducing the size of the fixed input/output arrays.
- Samples of the voltage are passed through the scaler block to 4 outputs *AnalogOutputX_SCALER*


This circuit is operated on a 40MHz clock, and the divisor is set to 4. That means that counter 1 will count once every 4 cycles, which is equivalent to a 10MHz clock. If the IOC is configured to a clock frequency other than 10MHz, then this clock needs to also be changed! It waits till a positive edge of pulse is detected, and it sums the analog input. Note that we are creating a digital pulse that feeds this scaler everytime we obtain a sample.


This loop takes 400 ticks of a 40MHz clock. A pulse is generated every 100us along with the sampled data. The delay of 400 ticks is important since we want to obtain a sample every 400 ticks to perform sum (integration.) Pay attention to the looptime. Sometimes after compilation it turns out to be larger than 400. Probably it is due to the space utilized on the FPGA. Further investigation is required. In the meantime, if the looptime is larger, try increasing the delay.

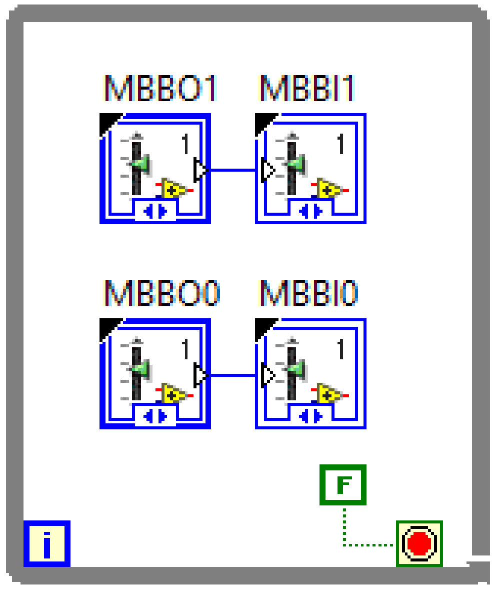
This frame is necessary to set the pulse input of the scaler IP

FPGA VI rules of thumb (5 / 6)

- Waveform FPGA support
  - Nheengatu supported array types : I64, I32, I16, I8, U64, U32, U16, U8, SGL (double not supported by FPGA and Fixed point not supported by C API generator)
  - Due to Labview limitations, only small arrays are supported in the FPGA VI
  - Just instantiate a control or indicator array of the upper types

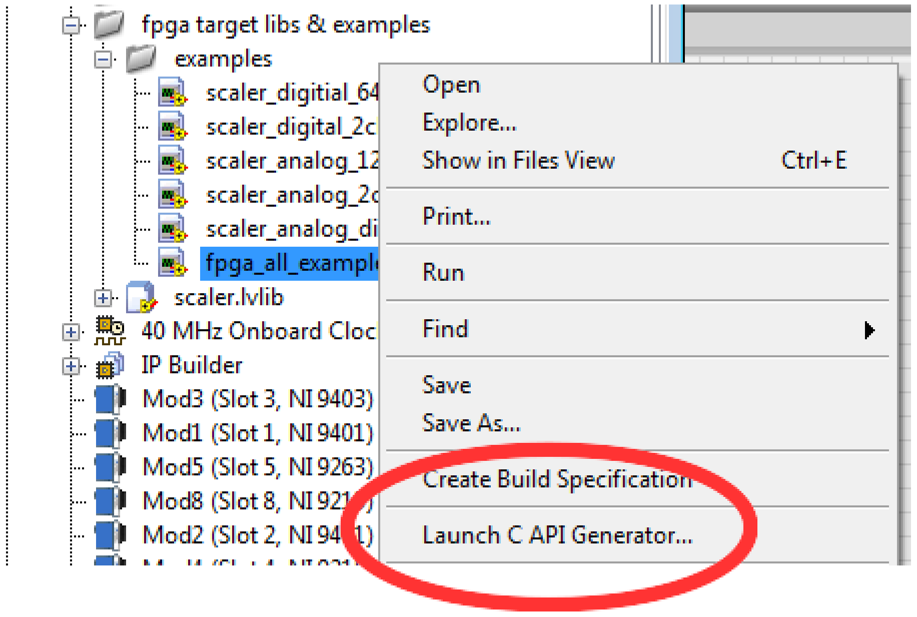
FPGA VI rules of thumb (6 / 6)

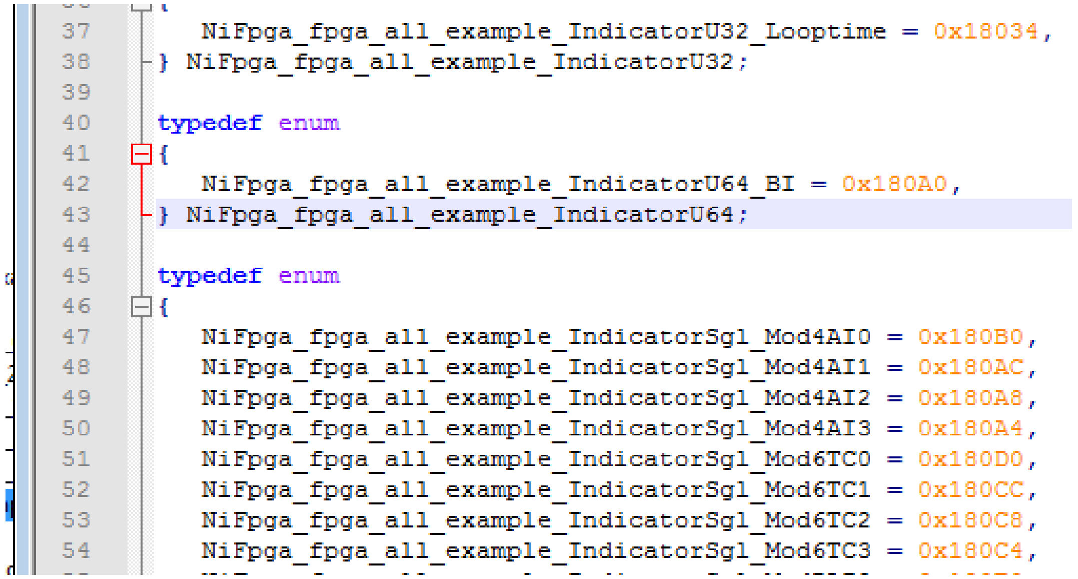
- MBBI / MBBO FPGA support
  - MBBI must use enums of presentation U16 and must be of type indicators
  - MBBO must use enums of presentation U16 and must be of type controls



Compiling your CRIO labview project

- Upon completion and generation of bitstream, the addresses of the FPGA VI must be generated
  - Rightclick FPGA VI > Launch C API generator
    - Save generated bitfile and header

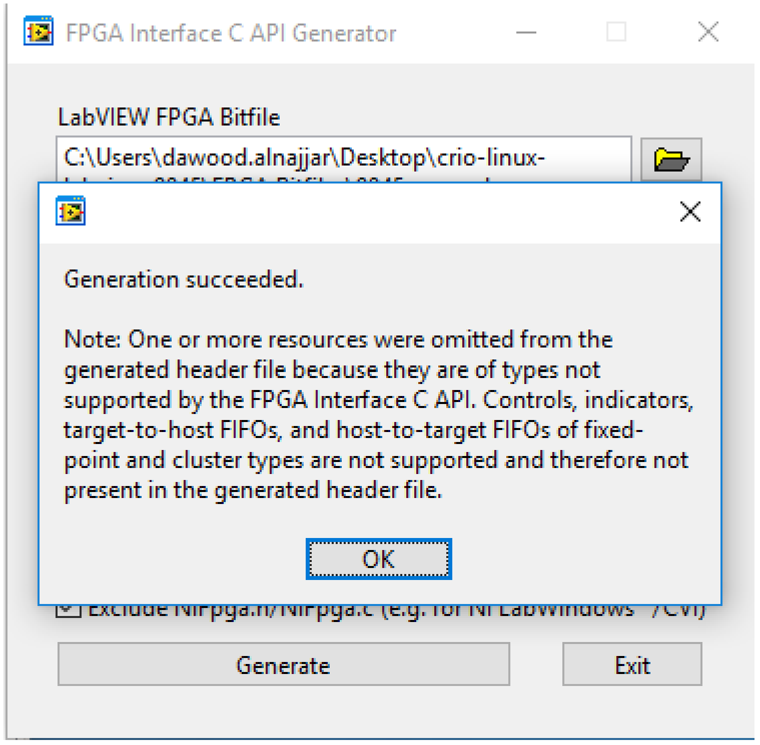




```
   NiFpga_fpga_all_example_IndicatorU32_Looptime = 0x18034,
} NiFpga_fpga_all_example_IndicatorU32;

typedef enum
{
   NiFpga_fpga_all_example_IndicatorU64_BI = 0x180A0,
} NiFpga_fpga_all_example_IndicatorU64;

typedef enum
{
   NiFpga_fpga_all_example_IndicatorSgl_Mod4AI0 = 0x180B0,
   NiFpga_fpga_all_example_IndicatorSgl_Mod4AI1 = 0x180AC,
   NiFpga_fpga_all_example_IndicatorSgl_Mod4AI2 = 0x180A8,
   NiFpga_fpga_all_example_IndicatorSgl_Mod4AI3 = 0x180A4,
   NiFpga_fpga_all_example_IndicatorSgl_Mod6TC0 = 0x180D0,
   NiFpga_fpga_all_example_IndicatorSgl_Mod6TC1 = 0x180CC,
   NiFpga_fpga_all_example_IndicatorSgl_Mod6TC2 = 0x180C8,
   NiFpga_fpga_all_example_IndicatorSgl_Mod6TC3 = 0x180C4,
```

Warning

Make sure that non of your indicators and controls are omitted by abiding with the data types defined above!



FPGA Interface C API Generator
LabVIEW FPGA Bitfile
OK
Exclude NiFpga.h/NiFpga.c (e.g. for NI LabWindows™/CVI)
Generate
Exit

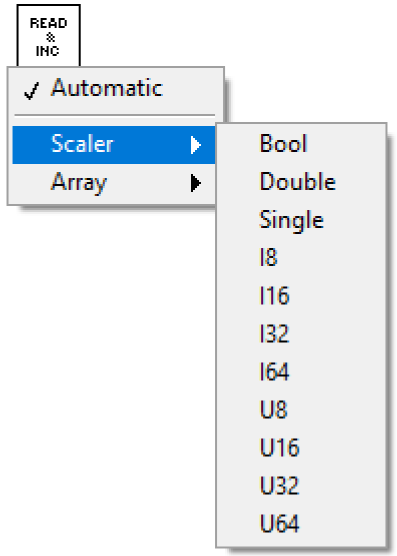
C:\Users\dawood.alnajjar\Desktop\crio-linux-

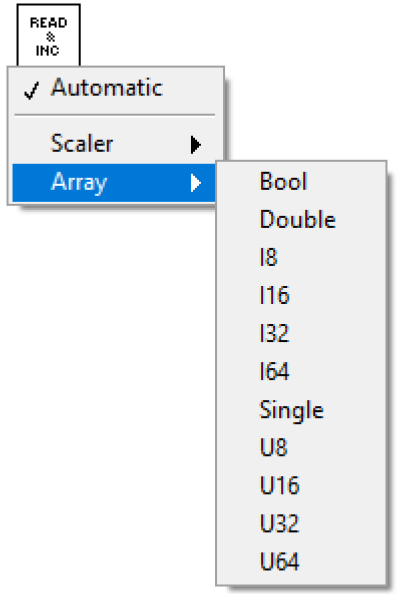
Generation succeeded.

Note: One or more resources were omitted from the generated header file because they are of types not supported by the FPGA Interface C API. Controls, indicators, target-to-host FIFOs, and host-to-target FIFOs of fixed-point and cluster types are not supported and therefore not present in the generated header file.

Labview RT VI Rules of thumb

- Ref : labviewRT_sm_example.vi
- No fixedpoint
- Developed lib VI/polymorphic VIs
  - IOC shared memory initialize.vi
  - IOC shared memory de-initialize.vi
  - SM read and increment.vi : Reads the variable from the shared memory
  - SM write and increment.vi : Writes the variable to the shared memory
- All VI MUST be chained with sm cluster input and output

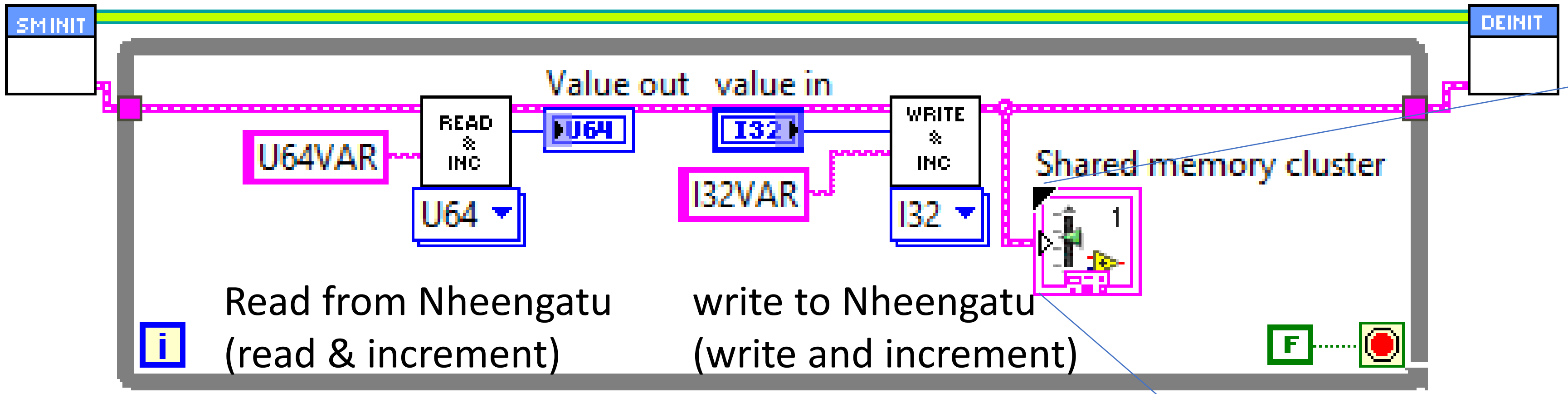




Labview-RT simple example

Shared memory initialize

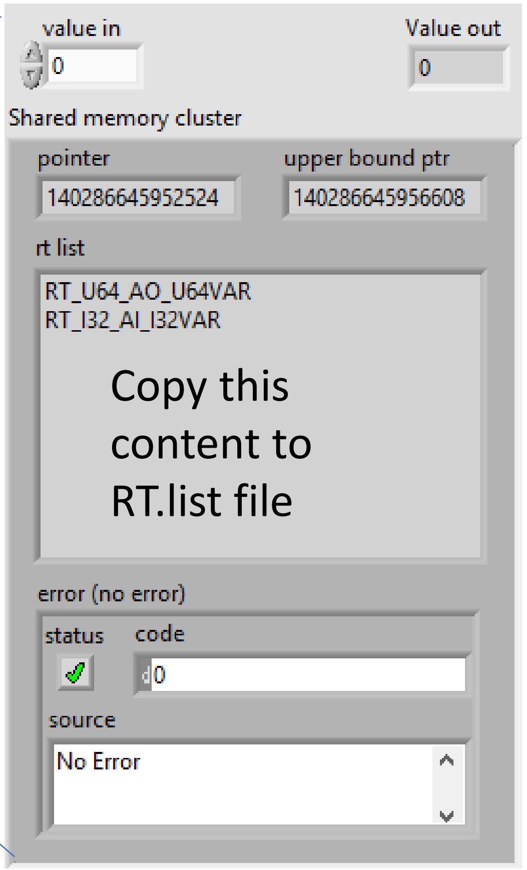
Shared memory de-initialize



Read from Nheengatu (read & increment)

write to Nheengatu (write and increment)

Cluster to generate RT.list and check errors if exist



value in: 0

Value out: 0

Shared memory cluster

pointer: 140286645952524

upper bound ptr: 140286645956608

rt list

```
RT_U64_AO_U64VAR
RT_I32_AI_I32VAR
```

Copy this content to RT.list file

error (no error)

status: ✓

code: d0

source: No Error

Preparing ini and substitutions (1 / 4)

<u>INI and substitutions can be generated manually (refer to any cfg.ini - have rules defined on top)</u>

- [Settings]
  - Use Shared Memory, Path, Shared Memory Path, Destination Crio IP, Bitfile Name, Signature, Shared Memory Size
- [BIAddresses], [BI0], [AO], [AI], [BO], [SCALERS], [WAVEFORMS], [MBBI], [MBBO]
- [FXP_XX]
  - Sign, Integer Word Length, Word Length
- [SCALERXX]
  - Done, Preset Values, Gate, OneShot, Enable, Number of Counters, Counters
- [WAVEFORMXX]
  - Type, Address, Size

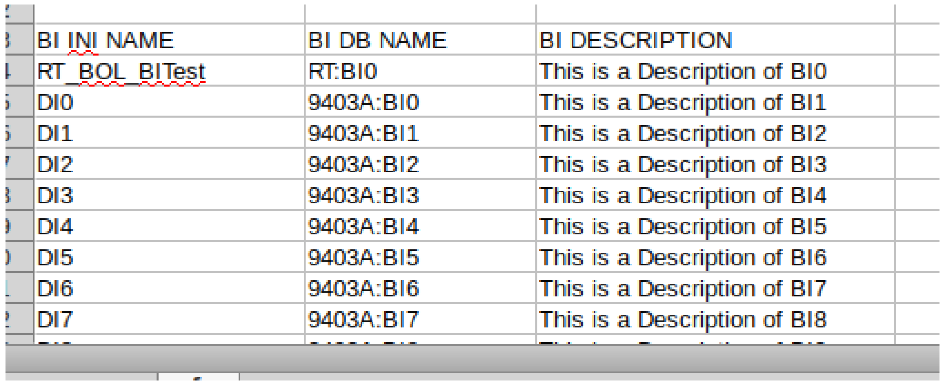
OR

Preparing ini and substitutions (2 / 4)

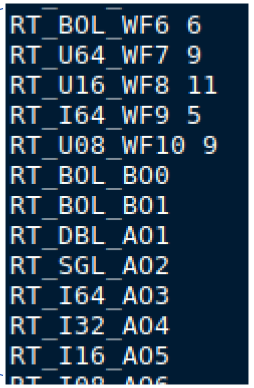
OR use generation script

- Repo : https://gitlab.cnpem.br/SOL/CRIO/crio-utils.git (contains examples)
- Use the script ./gen_cfg.py to process the header file. You may need to generate the RT file if labview RT is used.
- Check README.md for more information
- RT Variables naming in RT.list file (automatically generated in the VI cluster)
  - Naming must follow this syntax : RT_<VARIABLETYPE>_<NAME>.
    - VARIABLETYPE can be DBL, SGL, U64, U32, U16, U08, I64, I32, I16, I08, MBI, MBO
    - NAME must start with AI, BI, AO, BO, WF (when not MBI, MBO)
    - In case of WF, write length of array following the NAME and a space
- Run script twice. Once to generate a configuration template, and once to use the filled template

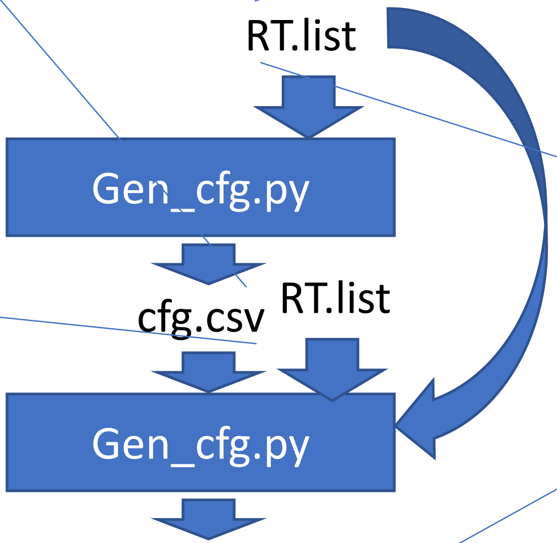
Preparing ini and substitutions (3 / 4)



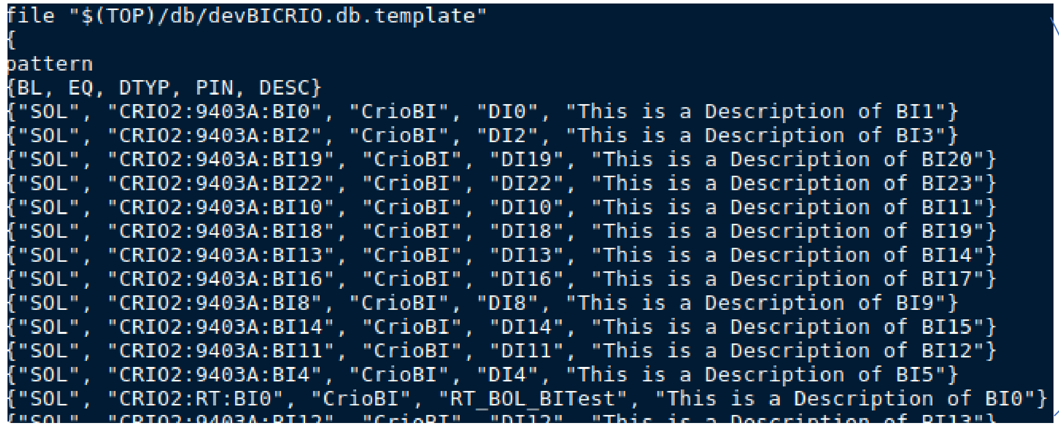
BI INI NAME	BI DB NAME	BI DESCRIPTION
RT_BOL_BITest	RT:BI0	This is a Description of BI0
DI0	9403A:BI0	This is a Description of BI1
DI1	9403A:BI1	This is a Description of BI2
DI2	9403A:BI2	This is a Description of BI3
DI3	9403A:BI3	This is a Description of BI4
DI4	9403A:BI4	This is a Description of BI5
DI5	9403A:BI5	This is a Description of BI6
DI6	9403A:BI6	This is a Description of BI7
DI7	9403A:BI7	This is a Description of BI8



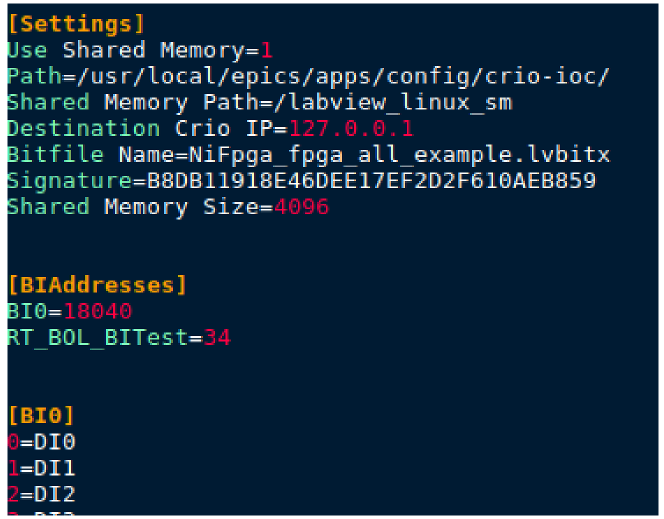
```
RT_BOL_WF6 6
RT_U64_WF7 9
RT_U16_WF8 11
RT_I64_WF9 5
RT_U08_WF10 9
RT_BOL_BO0
RT_BOL_BO1
RT_DBL_AO1
RT_SGL_AO2
RT_I64_AO3
RT_I32_AO4
RT_I16_AO5
```



Ready to go Cfg.ini, Substitutions, autosave & initial values!



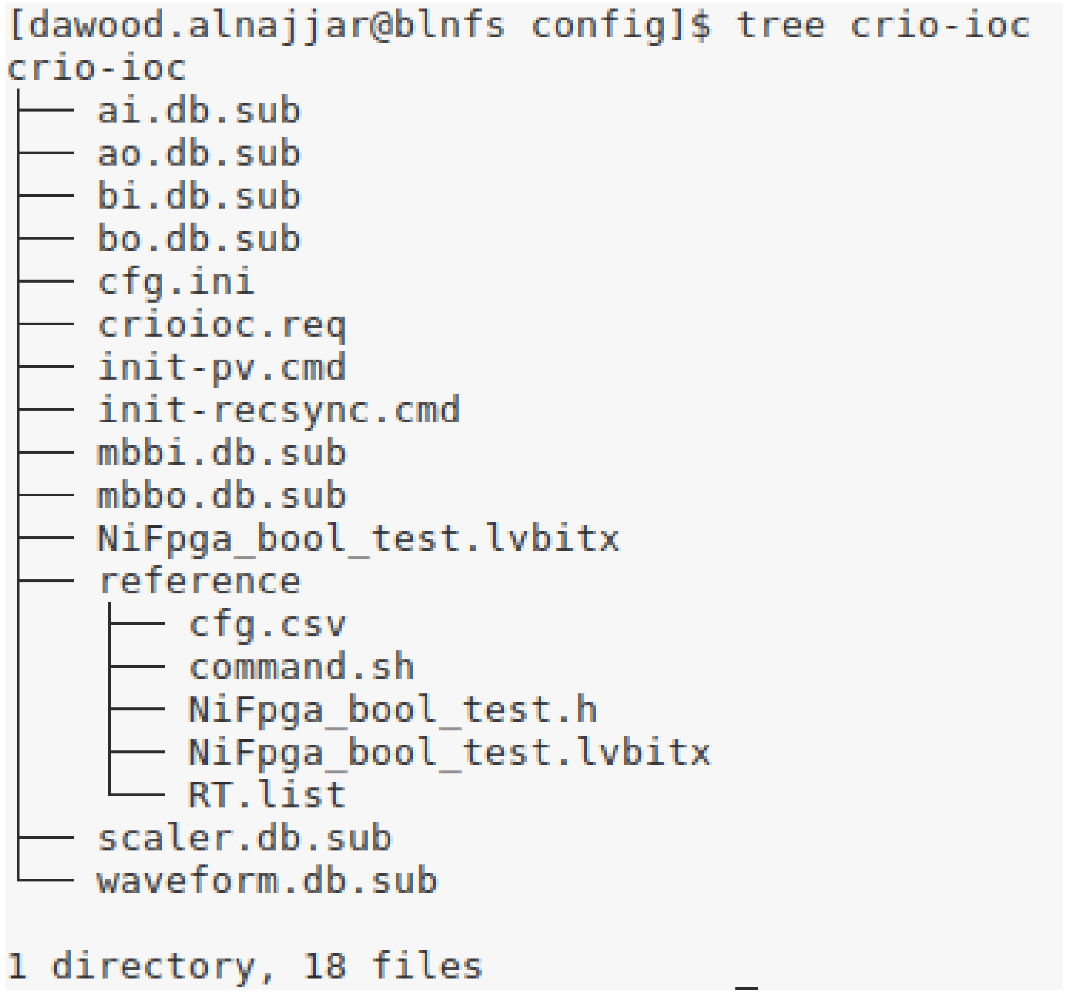
```
file "$(TOP)/db/devBICRIO.db.template"
{
pattern
{BL, EQ, DTYP, PIN, DESC}
{"SOL", "CRIO2:9403A:BI0", "CrioBI", "DI0", "This is a Description of BI1"}
{"SOL", "CRIO2:9403A:BI2", "CrioBI", "DI2", "This is a Description of BI3"}
{"SOL", "CRIO2:9403A:BI19", "CrioBI", "DI19", "This is a Description of BI20"}
{"SOL", "CRIO2:9403A:BI22", "CrioBI", "DI22", "This is a Description of BI23"}
{"SOL", "CRIO2:9403A:BI10", "CrioBI", "DI10", "This is a Description of BI11"}
{"SOL", "CRIO2:9403A:BI18", "CrioBI", "DI18", "This is a Description of BI19"}
{"SOL", "CRIO2:9403A:BI13", "CrioBI", "DI13", "This is a Description of BI14"}
{"SOL", "CRIO2:9403A:BI16", "CrioBI", "DI16", "This is a Description of BI17"}
{"SOL", "CRIO2:9403A:BI8", "CrioBI", "DI8", "This is a Description of BI9"}
{"SOL", "CRIO2:9403A:BI14", "CrioBI", "DI14", "This is a Description of BI15"}
{"SOL", "CRIO2:9403A:BI11", "CrioBI", "DI11", "This is a Description of BI12"}
{"SOL", "CRIO2:9403A:BI4", "CrioBI", "DI4", "This is a Description of BI5"}
{"SOL", "CRIO2:RT:BI0", "CrioBI", "RT_BOL_BITest", "This is a Description of BI0"}
```



```
[Settings]
Use Shared Memory=1
Path=/usr/local/epics/apps/config/crio-ioc/
Shared Memory Path=/labview_linux_sm
Destination Crio IP=127.0.0.1
Bitfile Name=NiFpga_fpga_all_example.lvbitx
Signature=B8DB11918E46DEE17EF2D2F610AEB859
Shared Memory Size=4096


[BIAddresses]
BI0=18040
RT_BOL_BITest=34


[BI0]
0=DI0
1=DI1
2=DI2
```

Preparing ini and substitutions (4 / 4)

Generate **reference** folder that has all files necessary to reproduce that specific setup



```
[dawood.alnajjar@blnfs config]$ tree crio-ioc
crio-ioc
├── ai.db.sub
├── ao.db.sub
├── bi.db.sub
├── bo.db.sub
├── cfg.ini
├── crioioc.req
├── init-pv.cmd
├── init-recsync.cmd
├── mbbi.db.sub
├── mbbo.db.sub
├── NiFpga_bool_test.lvbitx
├── reference
│   ├── cfg.csv
│   ├── command.sh
│   ├── NiFpga_bool_test.h
│   ├── NiFpga_bool_test.lvbitx
│   └── RT.list
├── scaler.db.sub
└── waveform.db.sub

1 directory, 18 files
```

./gen_cfg.py

```
usage: gen_cfg.py [-h] [-u] [-d DST] [-p PATH] [-s SRC] [--binum BINUM]
          [--extract] [--ip IP] [--smfname SMFNAME] [--smsize SMSIZE]
          [--aikey AIKEY] [--aokey AOKEY] [--bokey BOKEY]
          [--bikey BIKEY] [--mbbikey MBBIKEY] [--mbbokey MBBOKEY]
          [--fxpkey FXPKEY] [--scalerkey SCALERKEY]
          [--waveformkey WAVEFORMKEY] [--beamline BEAMLINE]
          [--bidtyp BIDTYP] [--aidtyp AIDTYP] [--bodtyp BODTYP]
          [--aodtyp AODTYP] [--mbbodtyp MBBODTYP]
          [--mbbidtyp MBBIDTYP] [--wfdtyp WFDTYP] [--crio CRIO]
          [--loc LOC] [--scalerdtyp SCALERDTYP] [--cfgcsv CFGCSV]
          [--refcsv] [--delimiter DELIMITER]
```

For help : ./gen_cfg.py -h

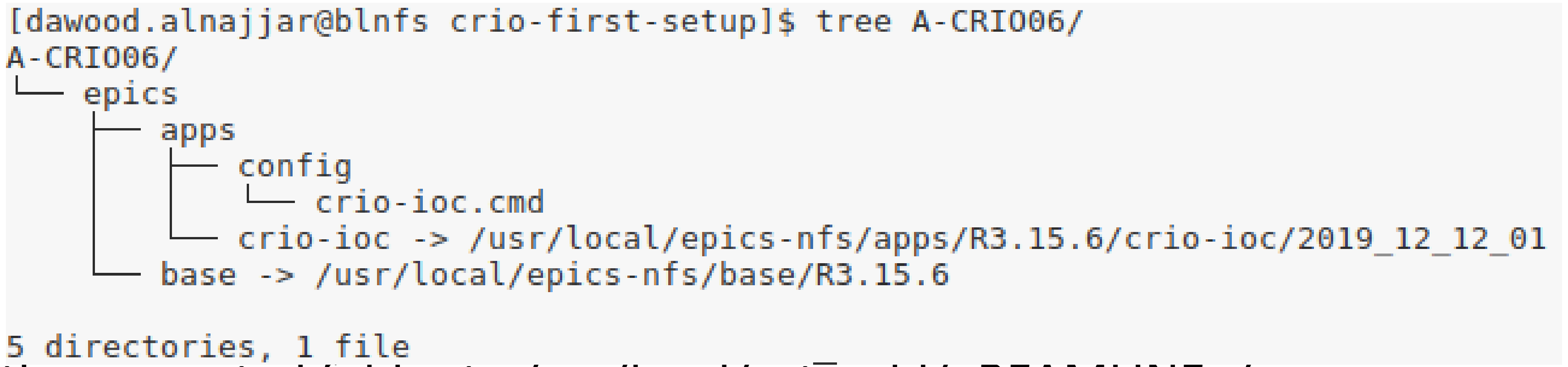
Prepare nfs beamline/device folder

- Create /usr/local/setup-bl/<BEAMLINE>/<LOCATION>-<CRIONAME>/epics folder
- Create a softlink to your epics version
  - ln –s /usr/local/epics-nfs/base/R3.15.6/ base
- Create apps folder
  - mkdir –p apps/config/crio-ioc && cd apps
- Softlink the latest CRIO-IOC
  - ln –s /usr/local/epics-nfs/apps/crio-ioc/2019_12_12_01/ crio-ioc (or the latest crio-ioc)
- Copy ini, substitutions and bitstream to nfs server /usr/local/setup-bl/<BEAMLINE>/<CRIONAME>/epics/apps/config/crio-ioc
- Copy crio-ioc.cmd to nfs server /usr/local/setup-bl/<BEAMLINE>/<CRIONAME>/epics/apps/config

OR

Prepare nfs beamline/device folder

- Use the crioSetupBlFolder.sh script installed in the /usr/bin folder of the nfs (also available in the https://gitlab.cnpem.br/SOL/Projetos/crio-first-setup.git repository.)(It generates symbolic links, and if scp is done, it will end up copying the real files instead of symlinks)
- usage: crioSetupBlFolder.sh <CRIO LOCATION> <CRIO POSTFIX> <CRIO IOC FOLDER NAME>
- Example : crioSetupBlFolders.sh A CRIO06 2019_12_12_01



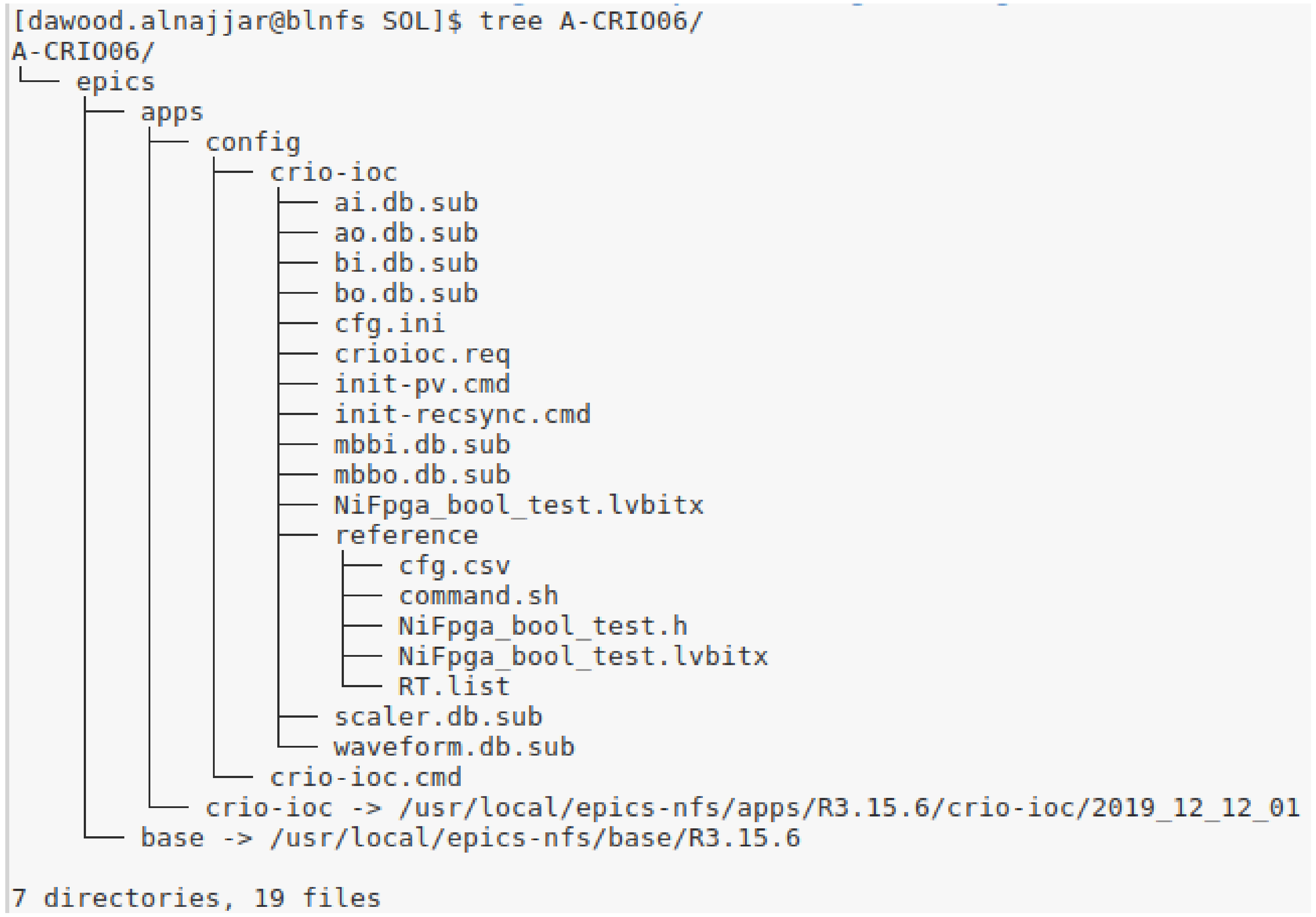
```
[dawood.alnajjar@blnfs crio-first-setup]$ tree A-CRIO06/
A-CRIO06/
└── epics
    ├── apps
    │   ├── config
    │   │   └── crio-ioc.cmd
    │   └── crio-ioc -> /usr/local/epics-nfs/apps/R3.15.6/crio-ioc/2019_12_12_01
    └── base -> /usr/local/epics-nfs/base/R3.15.6

5 directories, 1 file
```

- Move the generated folder to /usr/local/setup-bl/<BEAMLINE>/.

crio-ioc.cmd - Generic file (use as is)

```
#!/usr/local/epics/apps/crio-ioc/bin/linux-x86_64/CRIO
epicsEnvSet("TOP","/usr/local/epics/apps/crio-ioc")
epicsEnvSet("EPICS_BASE","/usr/local/epics-nfs/base/R3.15.6")
epicsEnvSet("IOC","iocCRIO")
epicsEnvSet("CONFIG","/usr/local/epics/apps/config/crio-ioc")
epicsEnvSet("AUTOSAVE","/opt/autosave")
epicsEnvSet("RECCASTER", "/usr/local/epics-
    nfs/apps/recsync/1.4_epics_3.15/client")
cd ${TOP}
dbLoadDatabase "dbd/CRIO.dbd"
CRIO_registerRecordDeviceDriver pdbbase
< "$(CONFIG)/init-recsync.cmd"
set_requestfile_path($(CONFIG))
set_savefile_path($(AUTOSAVE))
set_pass1_restoreFile("crioioc.sav", "")
crioSupSetup("${CONFIG}/cfg.ini" , 1)
## Load record instances
cd ${TOP}/iocBoot/${IOC}
dbLoadTemplate "${CONFIG}/bi.db.sub"
dbLoadTemplate "${CONFIG}/bo.db.sub"
dbLoadTemplate "${CONFIG}/ai.db.sub"
dbLoadTemplate "${CONFIG}/ao.db.sub"
dbLoadTemplate "${CONFIG}/scaler.db.sub"
dbLoadTemplate "${CONFIG}/waveform.db.sub"
dbLoadTemplate "${CONFIG}/mbbi.db.sub"
dbLoadTemplate "${CONFIG}/mbbo.db.sub"
iocInit
< "$(CONFIG)/init-pv.cmd"
create_monitor_set("crioioc.req", 1, "")
dbl
```

NFS setup-bl folder final structure



```
[dawood.alnajjar@blnfs SOL]$ tree A-CRIO06/
A-CRIO06/
└── epics
    ├── apps
    │   ├── config
    │   │   ├── crio-ioc
    │   │   │   ├── ai.db.sub
    │   │   │   ├── ao.db.sub
    │   │   │   ├── bi.db.sub
    │   │   │   ├── bo.db.sub
    │   │   │   ├── cfg.ini
    │   │   │   ├── crioioc.req
    │   │   │   ├── init-pv.cmd
    │   │   │   ├── init-recsync.cmd
    │   │   │   ├── mbbi.db.sub
    │   │   │   ├── mbbo.db.sub
    │   │   │   ├── NiFpga_bool_test.lvbitx
    │   │   │   ├── reference
    │   │   │   │   ├── cfg.csv
    │   │   │   │   ├── command.sh
    │   │   │   │   ├── NiFpga_bool_test.h
    │   │   │   │   ├── NiFpga_bool_test.lvbitx
    │   │   │   │   └── RT.list
    │   │   │   ├── scaler.db.sub
    │   │   │   └── waveform.db.sub
    │   │   └── crio-ioc.cmd
    │   └── crio-ioc -> /usr/local/epics-nfs/apps/R3.15.6/crio-ioc/2019_12_12_01
    └── base -> /usr/local/epics-nfs/base/R3.15.6

7 directories, 19 files
```

Setting up your CRIO – Linux

- Ssh to your CRIO
  - ssh admin@<CRIOIP>
  - passwd <sol legacy password>
  - opkg update
  - opkg install git
  - git clone https://gitlab.cnpem.br/SOL/Projetos/crio-first-setup.git
  - cd crio-first-setup
  - ./crioFirstSetup.sh (PBIS is not supported yet by CRIO)
    - Will create a default account with username <SOL> and password <sol legacy password>
- Now log out and log back in with the username SOL

Running the CRIO IOC with your configuration

- From CRIO run the following command and you are ready to go!
  - /usr/local/epics/apps/config/crio-ioc.cmd
- Or run the iocs script to run in background as follows
  - iocs start

Limitations

- Naming permitted characters [a-zA-Z0-9_]
- Binary inputs are limited to 64 inputs
- Moving U64, I64 (64-bits) from LabVIEW-RT to EPICS also is lossy since these variables are converted to double (52 bits precision)
- NI LabVIEW FPGA permitted array sizes (not necessary this is synthesizable, max array size tested U32 – 64 elements)
  - I8/U8 : 8188 elements
  - I16/U16: 4096 elements
  - I32/U32: 2047 elements
  - I64/U64: 1023 elements
- Synthesizable is ~75 elements

CRIO Tips

- Seems like controls default values do not work, and they are always set to 0
  - Numerical constants can be used
  - LabviewRT can be used to initialize the bitstream
  - IOC db file can be used to initialize
- The temperature module NI 9213 produces voltage readings, and this data needs to be processed.
  - Move this data to labviewRT and process it there. The temperature is then read from labviewRT by EPICS
- Large arrays on the FPGA can be moved to Labview RT using a DMA FIFO, and then passed to EPICS using waveform record