Hardware Triggered Scanning: From the Classroom to the Beamline

Philip Taylor, Emma Arandjelovic

Observatory Sciences Limited

Overview

This topic will cover the control of real hardware to perform scans, including:

- Issues related to motion control
- EPICS IOC prerequisites
- How to deploy Malcolm on beamlines
- Triggering setup for the ‘PMAC as master’ scenario

Motion Control

- Setup each Geobrick with the standard DLS config
- Make sure to include Prog1 (trajectory scan) and Prog10 (co-ordinate system) motion programs
- Take care to limit the maximum velocity and acceleration for programmed linear moves:
  - ixx16 – max velocity (counts / msec)
  - ixx17 – max acceleration (counts / $msec^2$)

EPICS Requirements

One or more motion IOCs should be setup to control the Geobricks involved in the scan

Instantiate:

1. *pmacController* template from the EPICS pmac module for each Geobrick with prefix BLxxP-MO-BRICK-nn
2. *pmacControllerTrajectory* template with the same prefix
3. *pmacCsController* template for each co-ordinate system (at least one)
4. *dls_pmac_asyn_motor* template for each motor axis
5. *dls_pmac_cs_asyn_motor* template for each cs axis

EPICS Requirements Cont.

- Malcolm uses the **VMAX, ACCL, ERES, EGU** and **OFF** design fields of the motor record
- VMAX is used to calculate an appropriate velocity for moving the motors to the start positions

EPICS Requirements Cont.

PandA interface

Recall: PandA box is also a source of AreaDetector data…

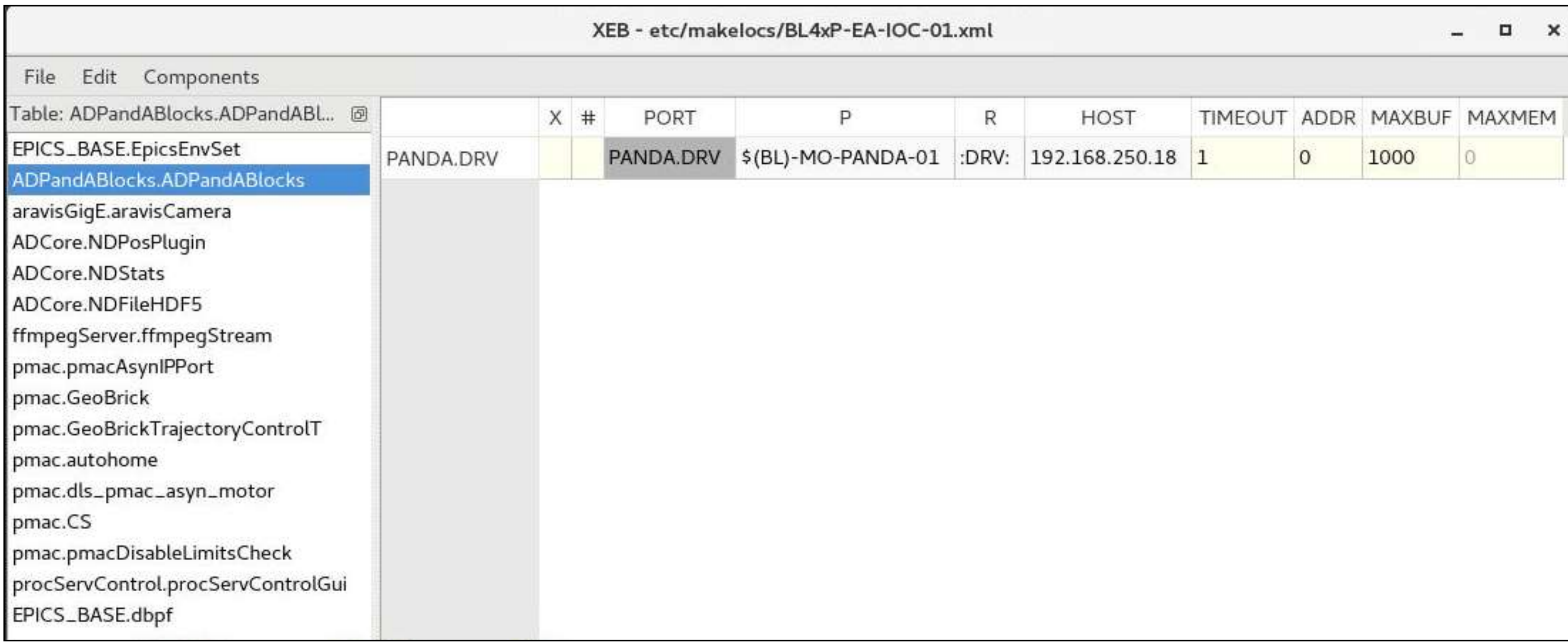
- Instantiate *ADPandABlocks* template from the ADPandaBlocks module with prefix BLxxI-MO-PANDA-nn:DRV:
- Setup the *NDPosPlugin* and *NDFileHDF5* AreaDetector plugins with prefixes BLxxI-MO-PANDA-nn:POS: and BLxxI-MO-PANDA-nn:HDF5:

Detectors

- Setup an additional AreaDetector plugin chain for each detector

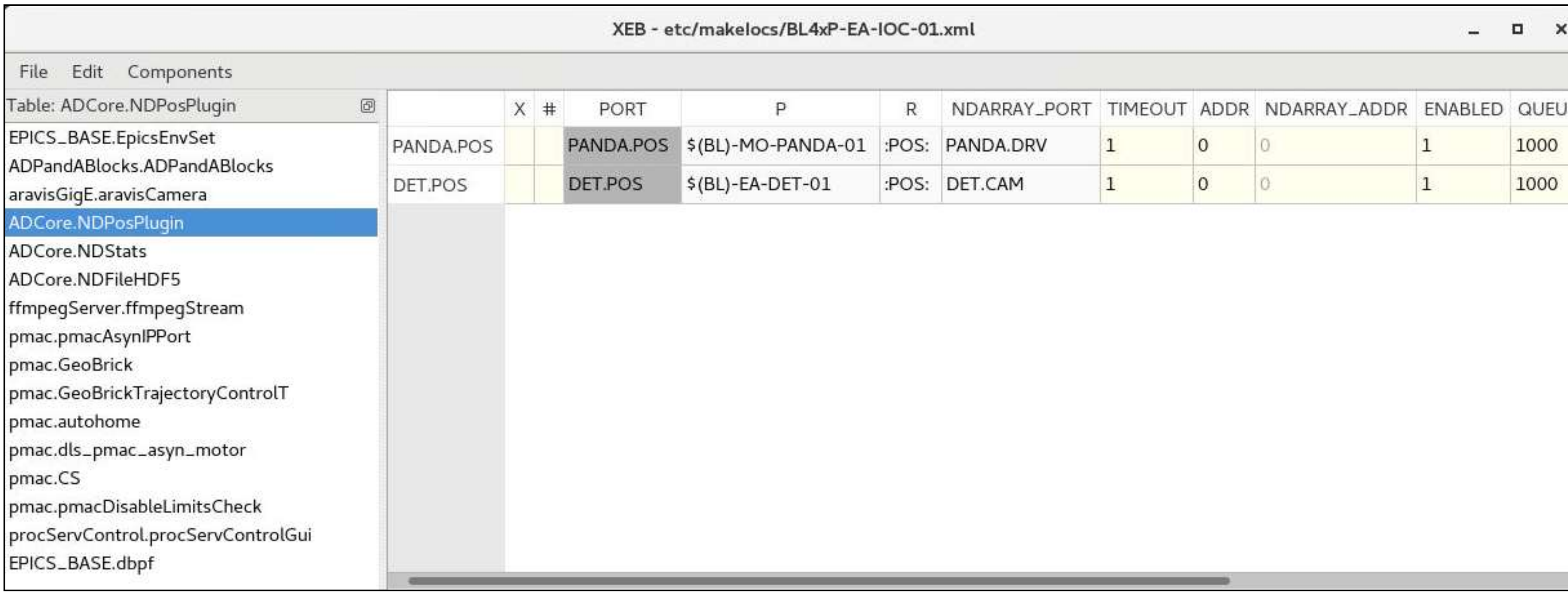
Example IOC Configuration

Typical iocbuilder file: ADPandABlocks setup



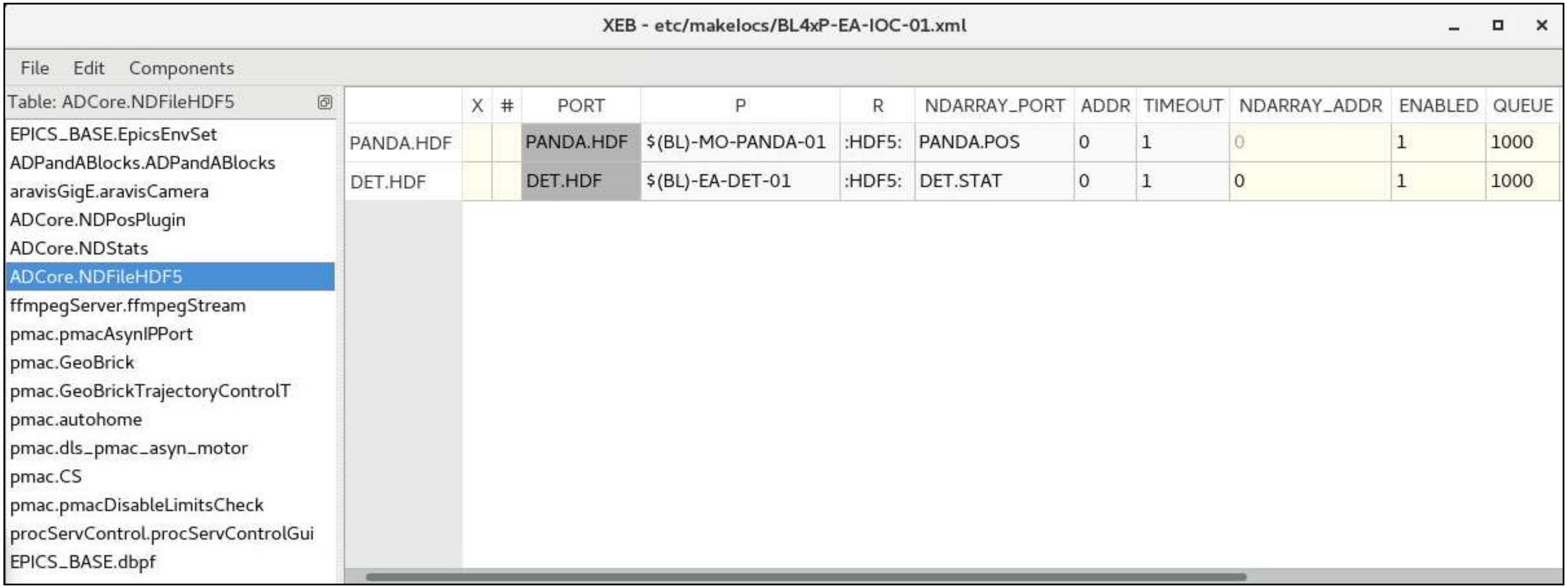
Example IOC Configuration

Typical iocbuilder file: NDPosPlugin setup



Example IOC Configuration

Typical iocbuilder file: NDFileHDF5 setup



Deploying Malcolm

1. Create a directory etc/malcolm in the BL IOC
2. Create a subdir blocks for the BL specific blocks
3. Create block definitions for each Geobrick and the top level scan block
4. Create the process definition: etc/malcolm/BLxxI-ML-MALC-01.yaml

See the PMAC tutorial: https://pymalcolm.readthedocs.io/en/latest/tutorials/pmac.html

Geobrick Blocks

- Construct using a ManagerController
- Use includes files to pull in the correct Blocks and Parts:

```
# Raw motor Blocks and their corresponding Parts
- pmac.includes.rawmotor_collection:
    mri: BLxxvimI-ML-STAGE-01:X
    pv_prefix: BLxxI-MO-MAP-01:STAGE:X
    scannable: stagex

# Co-ordinate system Block and its corresponding Part
- pmac.includes.cs_collection:
    mri_prefix: $(mri_prefix)
    pv_prefix: $(pv_prefix)
    cs: 1

# Trajectory scan and status Blocks and their corresponding Parts
- pmac.includes.trajectory_collection:
    mri_prefix: $(mri_prefix)
    pv_prefix: $(pv_prefix)
```

Geobrick Blocks

1. Locate the .yaml file for the Malcolm process running on the test rig
2. Drill down to find the block definition for the Geobrick and inspect it
3. Continue drilling down through the include statements to find the YAML for the motor records. What additional (runtime) fields of the motor record are accessed by Malcolm?

Scan Block

- As in the scanning tutorial, this uses a RunnableController
- We add an initial_design parameter to pass to the controller *

```
- builtin.parameters.string:
    name: config_dir
    description: Where to store saved configs

- builtin.parameters.string:
    name: initial_design
    description: Initial design to load for the scan

- scanning.controllers.RunnableController:
    mri: $(mri_prefix)
    config_dir: $(config_dir)
    description: Hardware triggered scan
    initial_design: $(initial_design)
```

> * This means multiple instances can be used, each with a different starting configuration

Scan Block Continued

We also need some same part definitions:

- LabelPart

  Provides a human readable label for the Block. 4 or 5 words that describe the science case for this scan
- SimultaneousAxesPart

  Defines the superset of all axes that can be supplied as axesToMove at configure()
- DatasetTablePart

  Reports the datasets that any detectors produce
- PmacChildPart

  Takes the generator passed to configure(), and produces trajectory scan points that can be passed down to a PMAC Block
- DetectorChildPart

  Controls a detector (a runnable child block with a datasets Attribute)

Malcolm Process Definition

BL/etc/malcolm/BLxxI-ML-MALC-01.yaml

```
#!/dls_sw/prod/common/python/RHEL7-x86_64/pymalcolm/4-0/malcolm/imalcolm.py

- builtin.defines.module_path:
    name: BLxxI
    path: $(yamldir)

- builtin.defines.string:
    name: config_dir
    value: /dls_sw/ixx/epics/malcolm

- BLxxI.blocks.brick01_block:
     mri_prefix: BLxxI-ML-BRICK-01
     pv_prefix: BLxxI-MO-BRICK-01
     config_dir: $(config_dir)
```

- Make it executable by providing the full path to imalcolm
- Define this directory as the BLxxI module
- Specify where designs are saved
- Instantiate some Geobrick blocks (to be created in the blocks subdir)

Malcolm Process Definition

etc/malcolm/BLxxP-ML-MALC-01.yaml cont…

```
- ADPandABlocks.blocks.
panda_runnable_block:
    mri_prefix: BLxxP-ML-PANDA-01
    pv_prefix: BLxxP-MO-PANDA-01
    hostname: 192.168.250.18
    config_dir: $(config_dir)

- BLxxP.blocks.pmac_master_scan_block:
    mri_prefix: BLxxP-ML-SCAN-01
    config_dir: $(config_dir)
    initial_design:
```

- Instantiate a block for the PandA (from the Malcolm ADPandABlocks module)
- Instantiate a scan block (to be created in the blocks subdir)

Malcolm Process Definition

etc/malcolm/BLxxP-ML-MALC-01.yaml cont…

```
# Scan the DLS redirector for IOCs to monitor, optional
- system.defines.redirector_iocs:
    name: iocs
    yamlname: $(yamlname)

# Define the ServerComms
- system.blocks.system_block:
    mri_prefix: $(yamlname)
    iocs: $(iocs)
    pv_prefix: $(yamlname)
    config_dir: $(config_dir)
```

- Define the comms interfaces (web server and pvAccess for GDA) using system_block

Malcolm Modules

- To use our newly created beamline specific blocks, we need to declare them
- To do this we make the blocks directory a *Python package* by creating __init__.py
- This is run by Python whenever the package is imported
- We also create an empty __init__.py in the top level <BL>/etc/malcolm directory

Block Module __init__.py

The __init__.py for the Blocks module needs to map Block names to corresponding .yaml files:

```
from malcolm.yamlutil import make_block_creator, check_yaml_names

# Create some Block definitions from YAML files
brick01_block = make_block_creator(__file__, "brick01_block.yaml")
another_block = make_block_creator(__file__, “another_block.yaml")
……….

# Expose all of the Block definitions, and nothing else
__all__ = check_yaml_names(globals())
```

Final Exercise

Create a new Malcolm instance from scratch to do a scan using the motion controller and PandA, but no detector, on the test rig.

We'll use the IOC that is already running on the testrig.

This IOC is used to control the PMAC and provides PandA AD plugins. It provides:

- A Geobrick controller template with PV prefix BLxxP-MO-BRICK-01
- Raw axes with PV prefixes BLxxP-MO-STAGE-01:{X,A}
- An ADPandABlocks template with PV prefix BLxxP-MO-PANDA-01

Code can be copied from:
https://pymalcolm.readthedocs.io/en/latest/tutorials/pmac.html

Final Exercise Part 1

1. Create the required directory structure in /scratch
2. Create a PMAC block in the blocks subdir. It needs:
   i. a *builtin.controllers.ManagerController*
   ii. one *pmac.includes.rawmotor_collection* per axis
   iii. a *pmac.includes.cs_collection*
   iv. a *pmac.includes.trajectory_collection*
3. Create a scan block in the blocks subdir. It needs:
   i. a *scanning.controllers.RunnableController*
   ii. an empty *scanning.parts.SimultaneousAxesPart*
   iii. a *scanning.parts.DatasetTablePart*
   iv. a *pmac.parts.PmacChildPart* for the PMAC
   v. a *scanning.parts.DetectorChildPart* for the detector and the PandA

Tip: Hard coding prefixes instead of passing them around as in the online tutorials keeps things a bit simpler for now!

Final Exercise Part 1

4. Create the process definition *BLxxP-ML-MALC-01.yaml*
   – Add a *builtin.defines.module_path* (set to *yamldir*) so that you can import your blocks.
   – Instantiate PMAC and scan blocks
   – Instantiate an *ADPandABlocks.blocks.panda_runnable_block*
   – Don’t forget the comms interfaces
   – Make it executable
5. Try to start up Malcolm and fix any errors
   – Make sure you modify the PV prefixes, host names etc to match your test rig!

Final Exercise Part 1

- Problems?
  - Are your __init__.py files correct?

etc/malcolm/blocks/__init__.py:

```
from malcolm.yamlutil import make_block_creator, check_yaml_names

brick01_block = make_block_creator(__file__, "brick01_block.yaml")
scan_block = make_block_creator(__file__, "scan_block.yaml")

__all__ = check_yaml_names(globals())
```

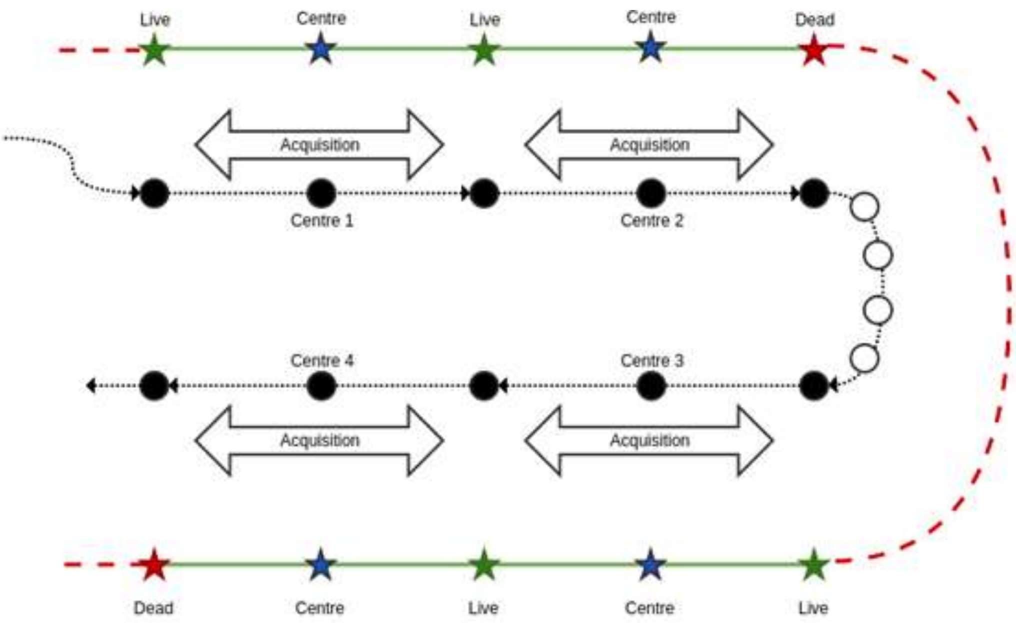
Final Exercise Part 2

6. Using the web GUI, modify the brick design to assign the motors to any two axes in the CS and save the design
7. Load the template_live_dead_framed_pcap design onto the PandA block
8. Setup the scan block:
   - Give a value to the label (e.g. “small stage tomography”)
   - Set SimultaneousAxes to the scannable names of the motors in the CS
   - Save the design using a similar name to the label
9. Run a scan from the web GUI or console to check it works:
   - Ensure the scan path is within the physical motor limits for your rig!
   - Suggest a duration of 0.005 for the CompoundGenerator to ensure a safe velocity
10. Set the initial_design in your process definition and restart

Check signals

Recall the trigger setup:

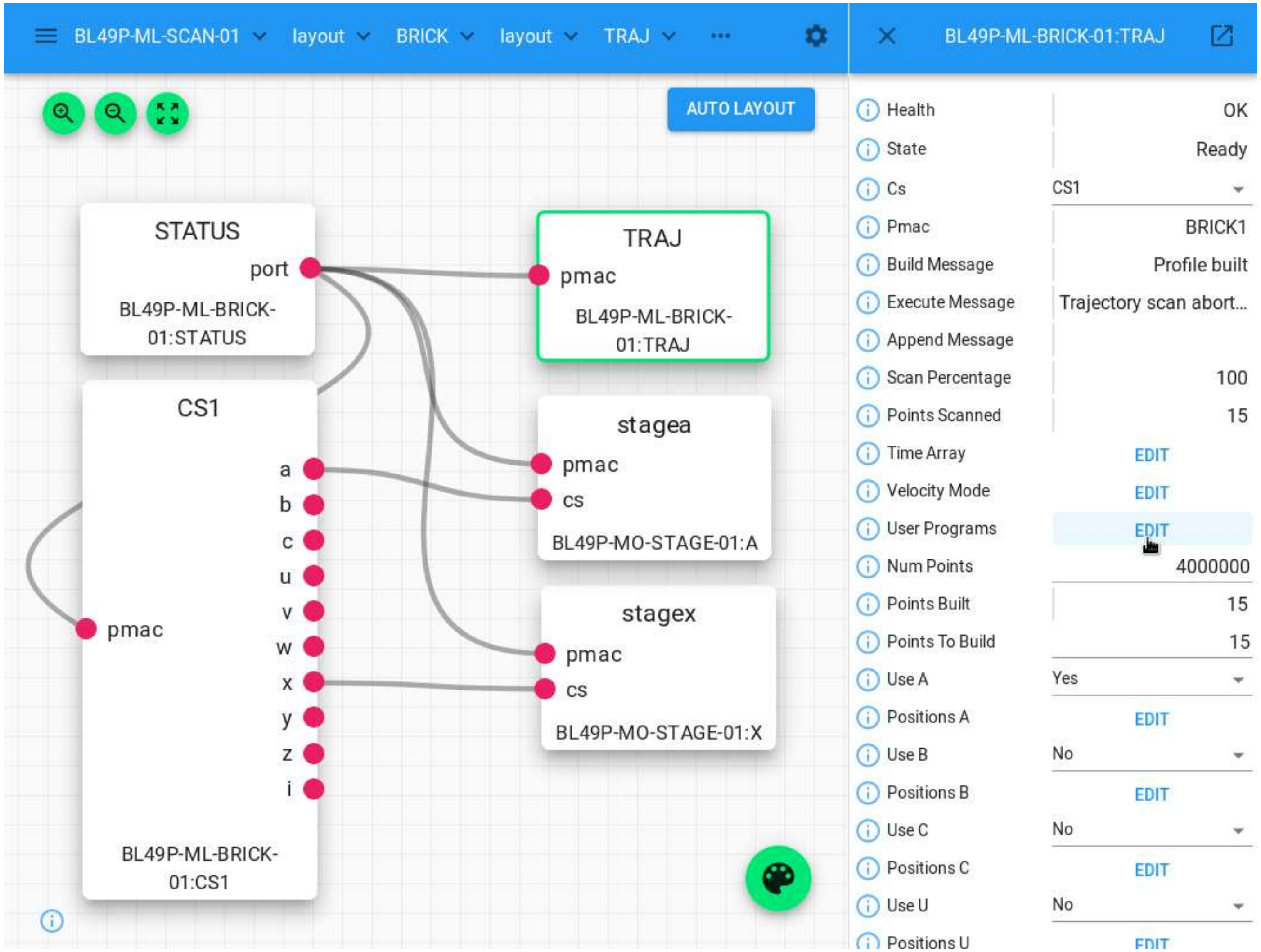
- 'Live' signal sent at the start of each point
- 'Dead' signal sent at the end of each point if it doesn't join the next one
- We want to capture the **average** positions at the end of every real frame



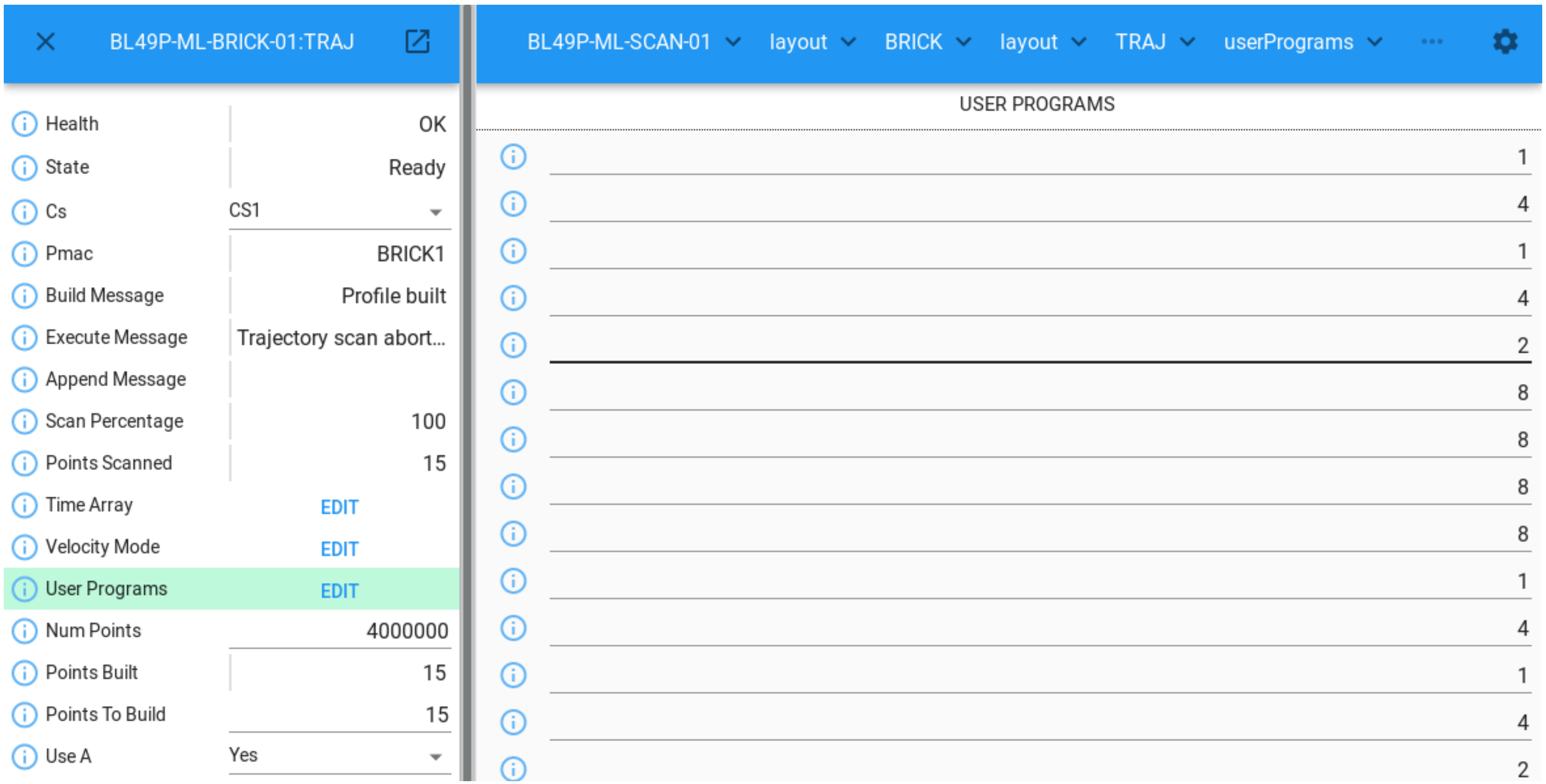
Mapping of triggers to their user programs:

```
NO_PROGRAM = 0 # Do nothing
LIVE_PROGRAM = 1 # GPIO123 = 1, 0, 0
DEAD_PROGRAM = 2 # GPIO123 = 0, 1, 0
MID_PROGRAM = 4 # GPIO123 = 0, 0, 1
ZERO_PROGRAM = 8 # GPIO123 = 0, 0, 0
```

Check signals



Check signals

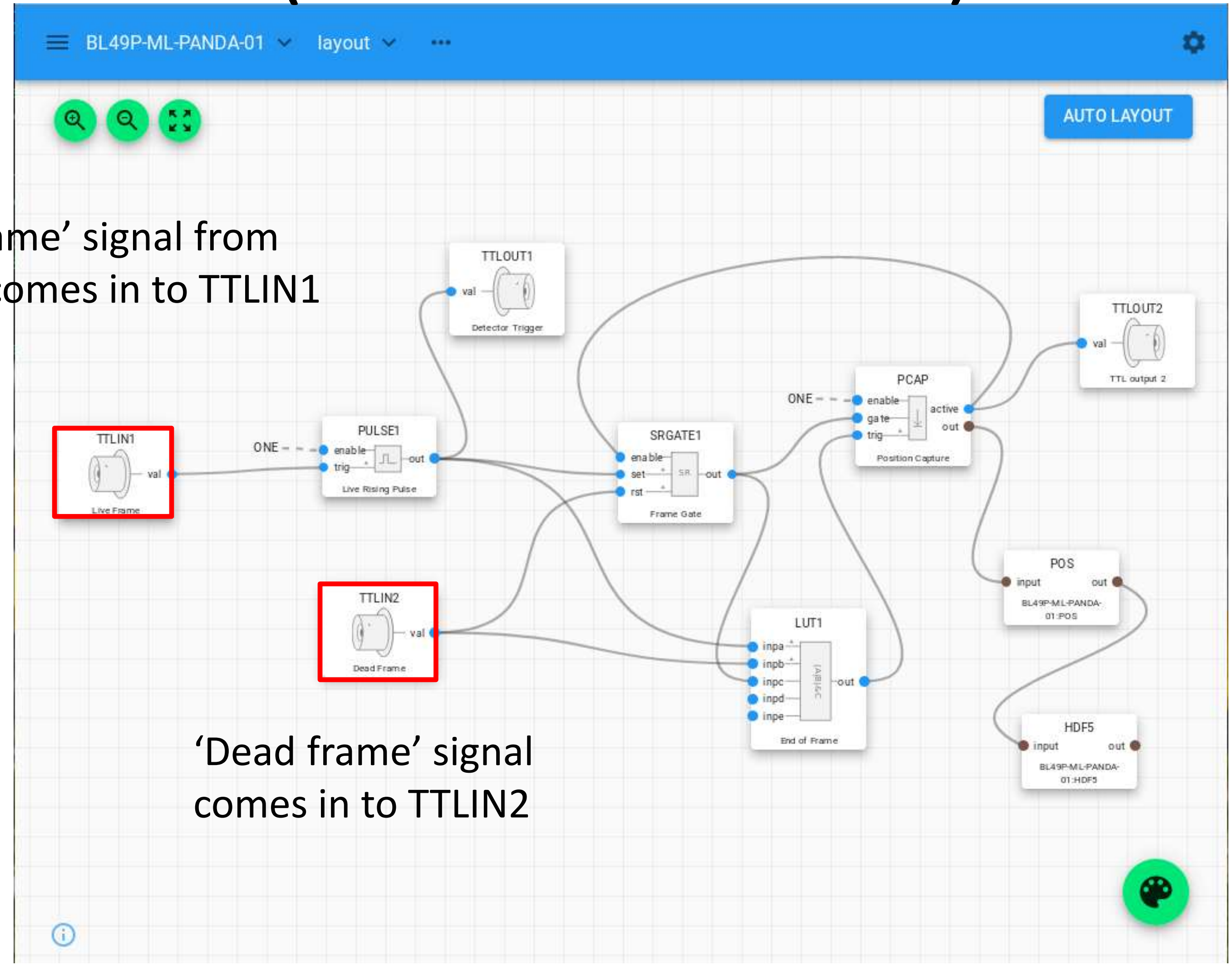


PMAC as Master v. PandA as Master

- There are two overarching designs to control a scan: PMAC or PandA as Master
- The Master is responsible for generating live and dead signals
- PMAC as Master is simpler, so start there
- Limited to 300 Hz
- If you need faster than this, go for PandA as Master

PandA Design (PMAC as master)

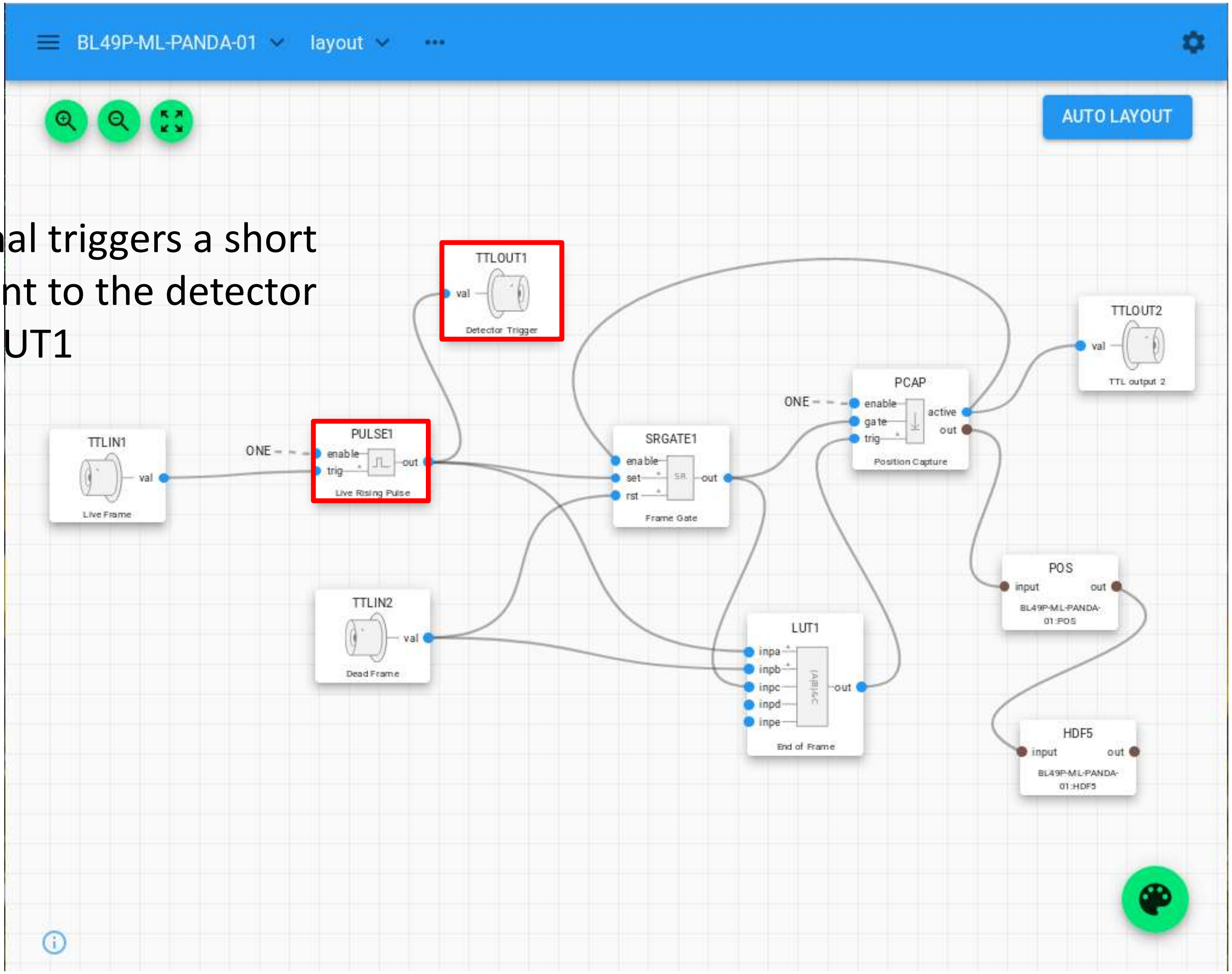
‘Live frame’ signal from PMAC comes in to TTLIN1



‘Dead frame’ signal comes in to TTLIN2

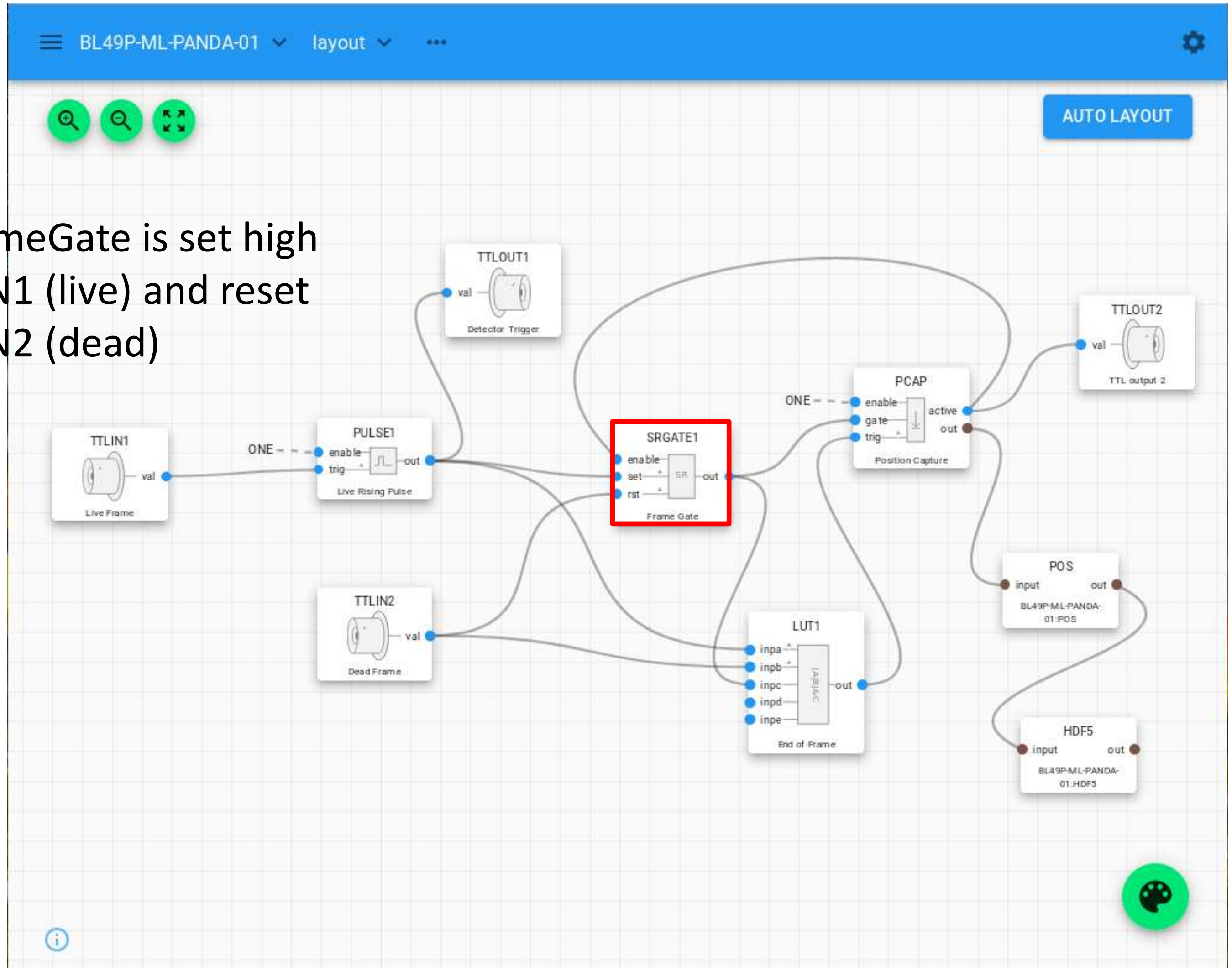
PandA Design

Live signal triggers a short pulse sent to the detector on TTLOUT1



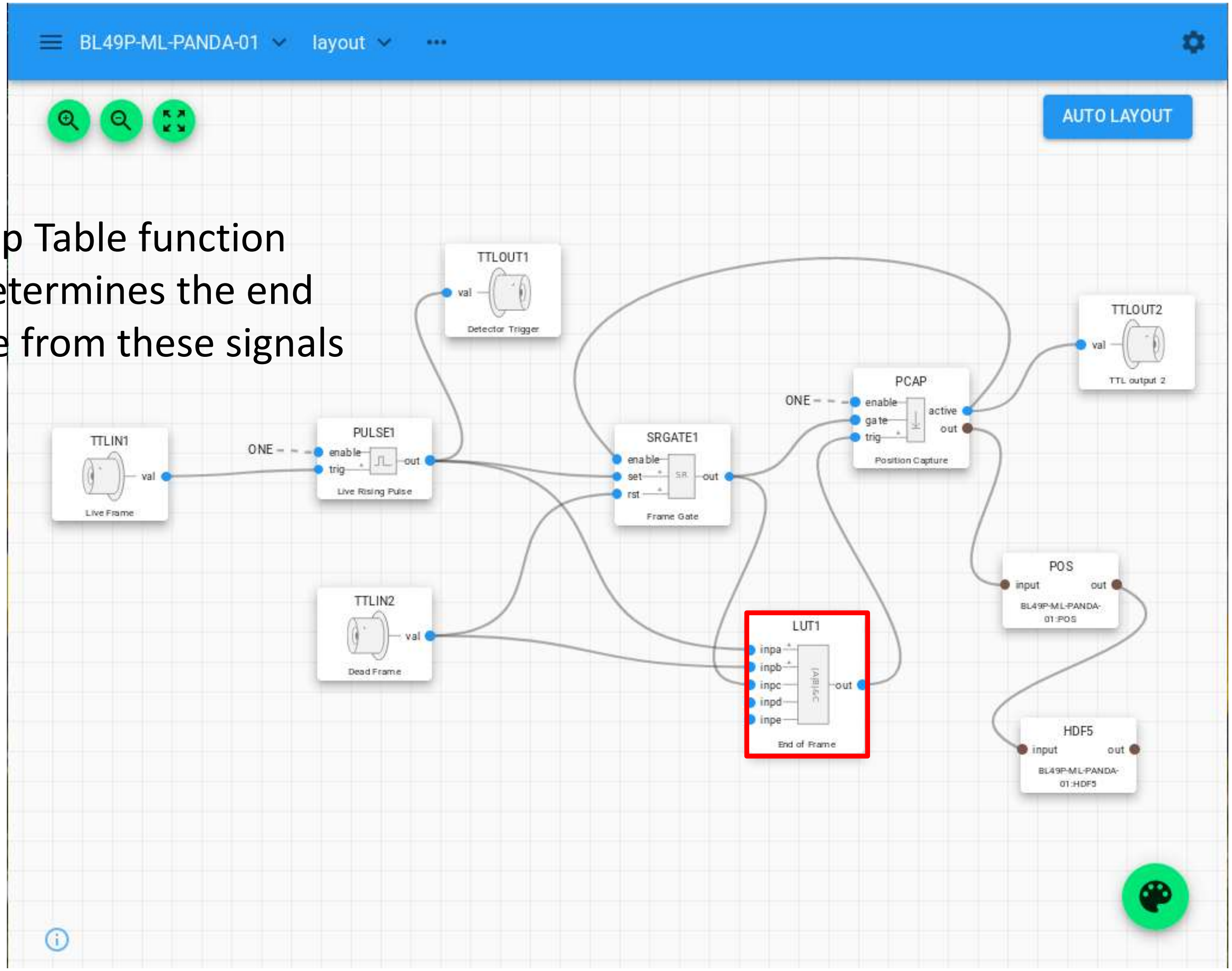
PandA Design

The FrameGate is set high by TTLIN1 (live) and reset by TTLIN2 (dead)



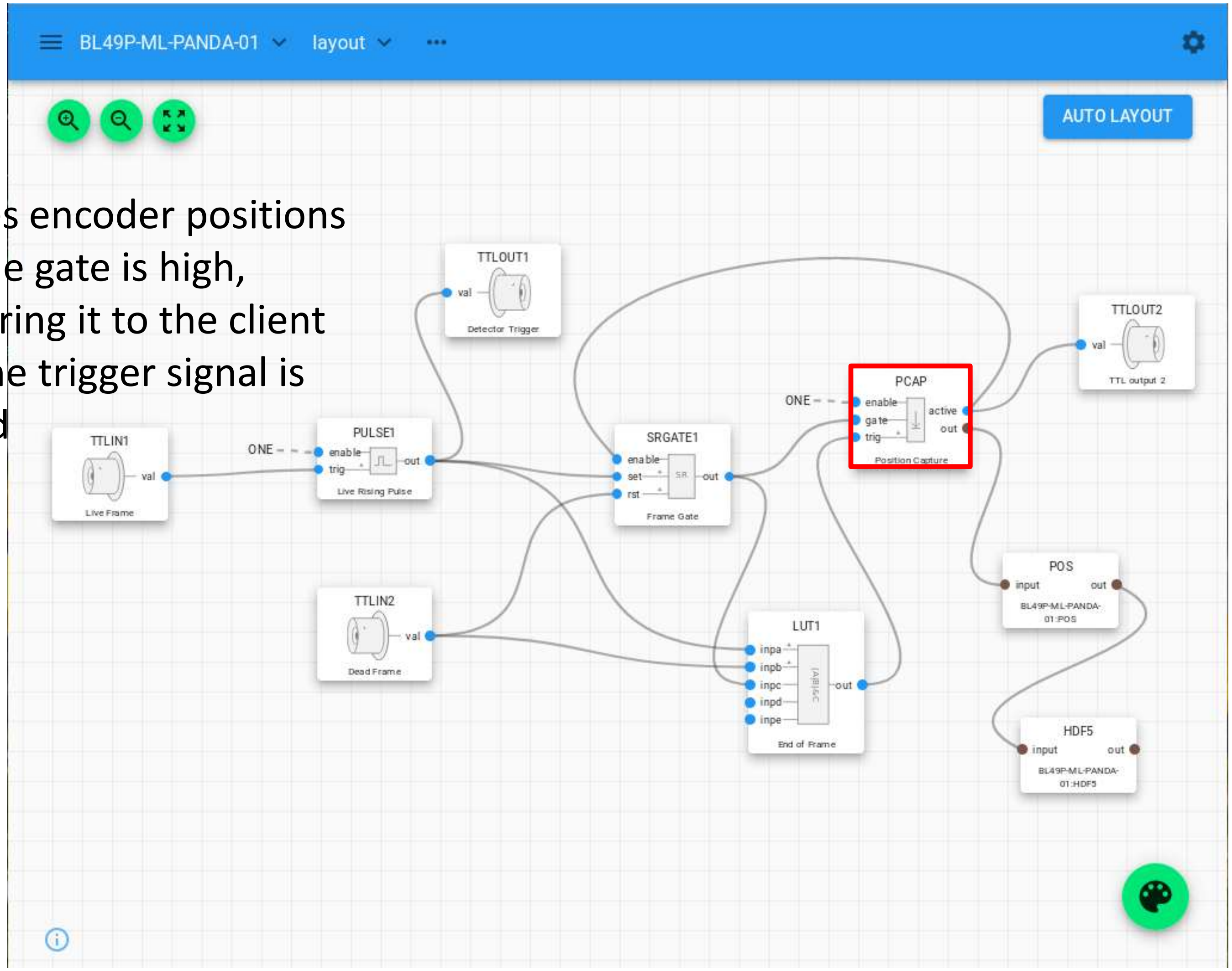
PandA Design

A Lookup Table function block determines the end of frame from these signals



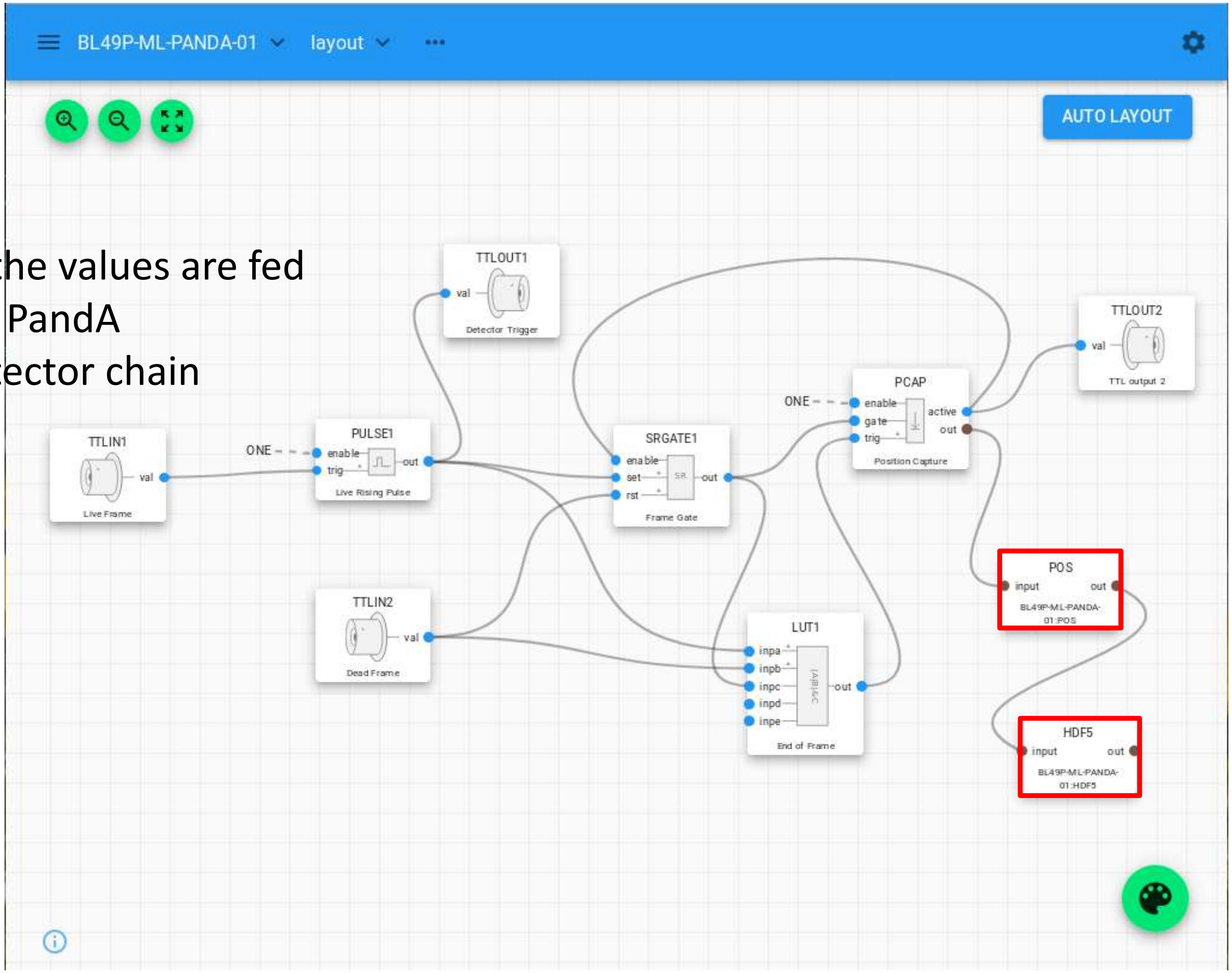
PandA Design

Averages encoder positions while the gate is high, transferring it to the client when the trigger signal is asserted

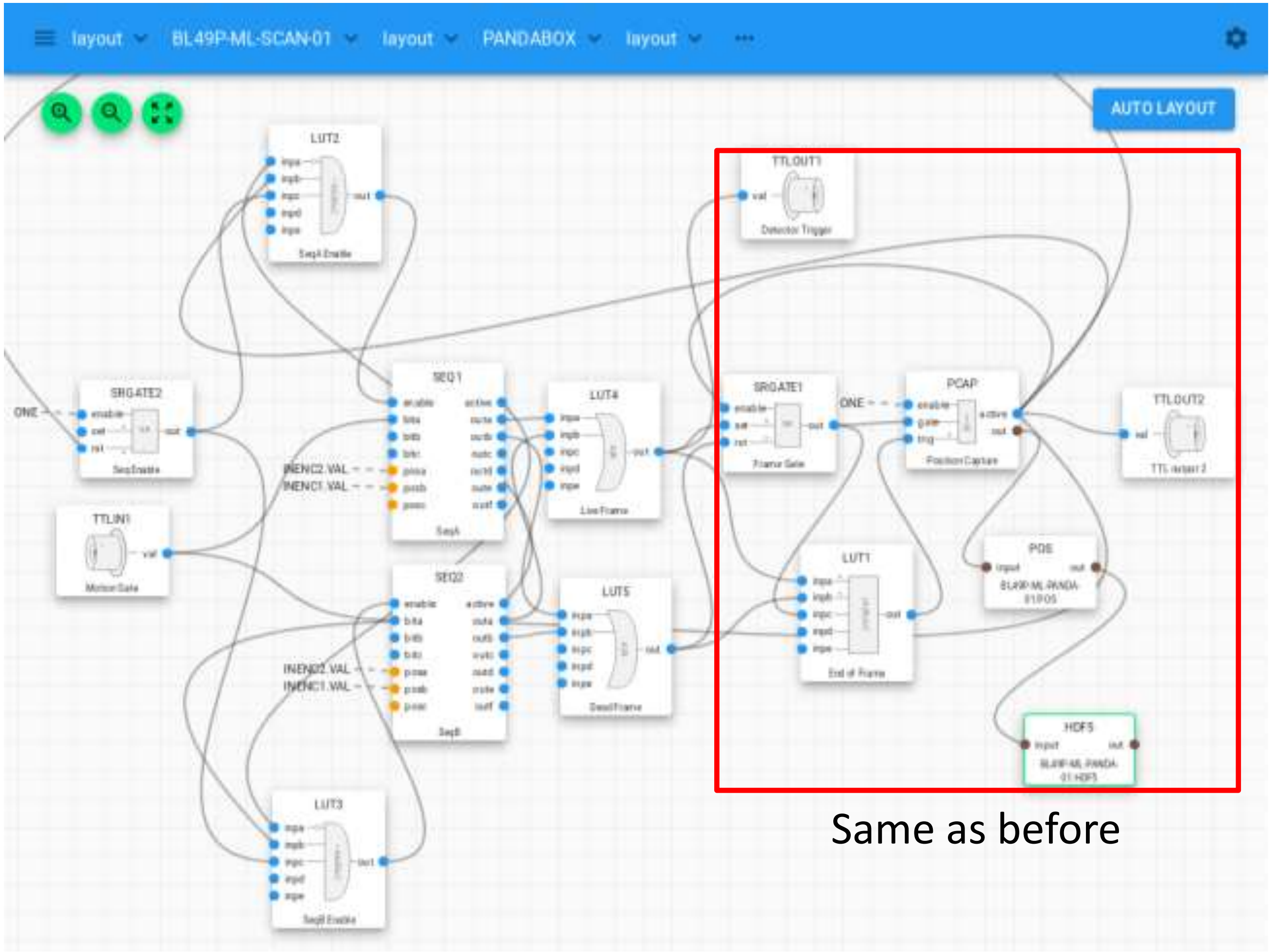


PandA Design

Finally, the values are fed into the PandA AreaDetector chain



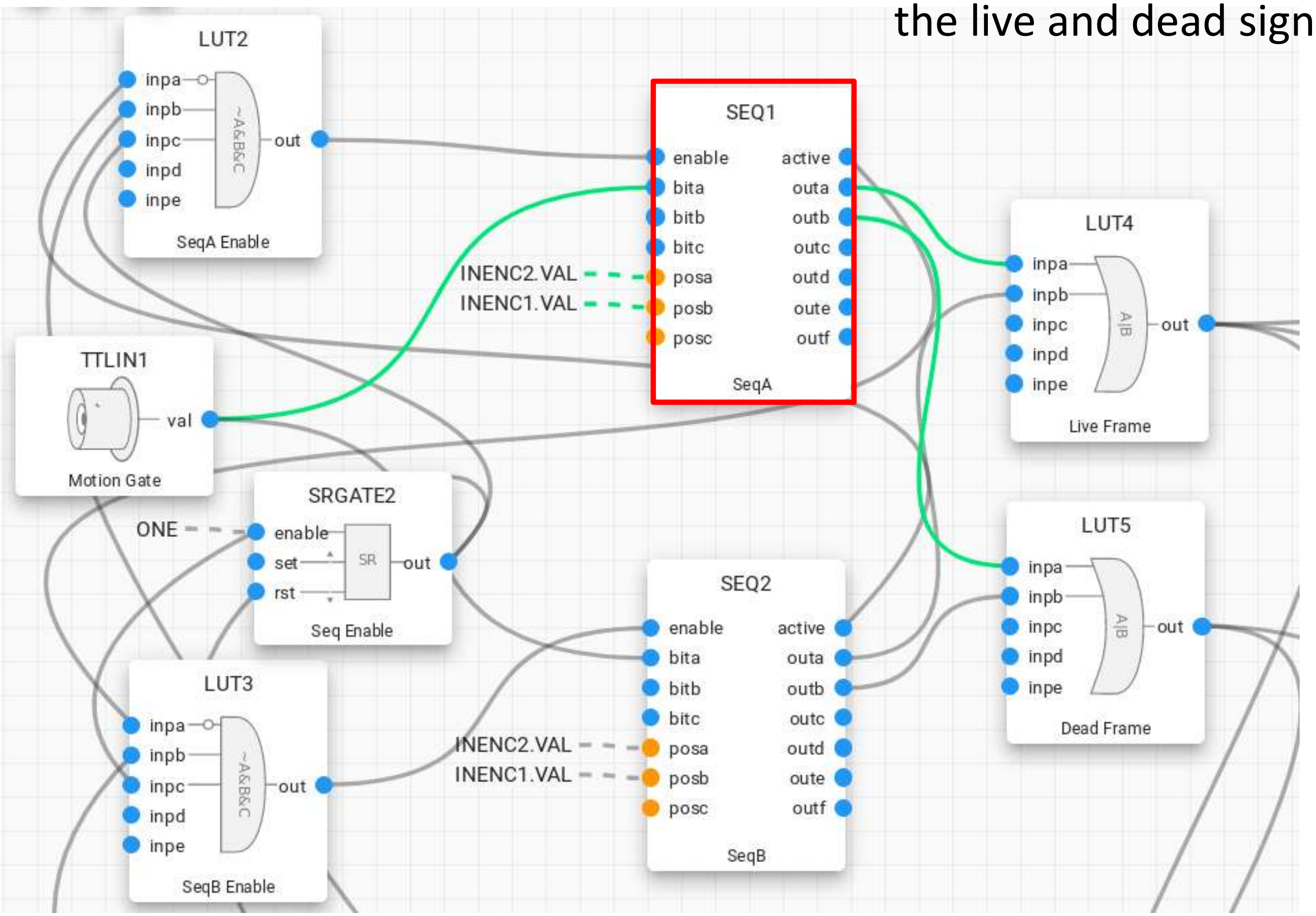
PandA as Master



Same as before

PandA as Master

Sequence table generates the live and dead signals



PandA as Master – seq table

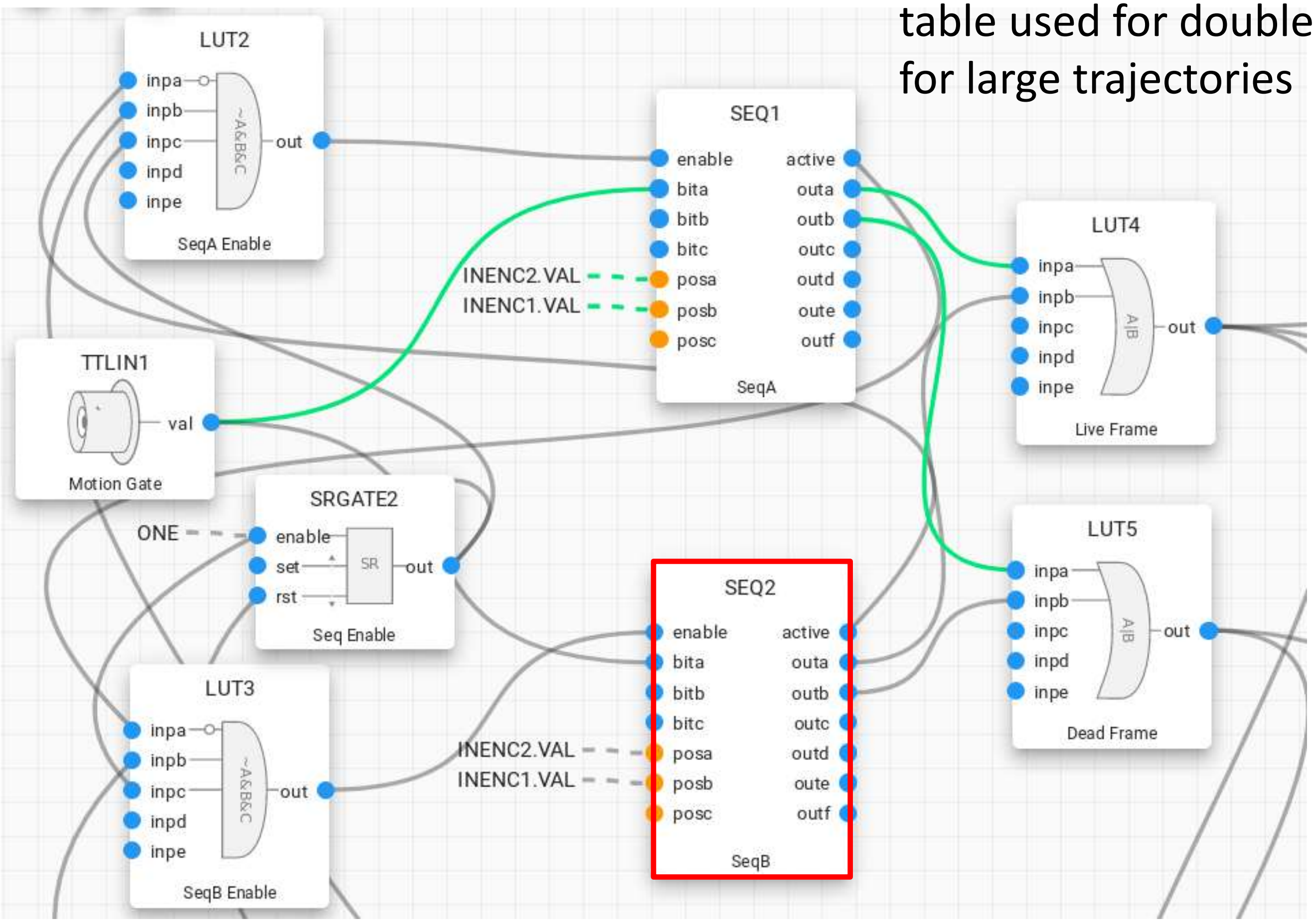
Live signal Dead signal

layout ⌄ BL49P-ML-SCAN-01 ⌄ layout ⌄ PANDABOX ⌄ layout ⌄ SEQ1 ⌄ table ⌄ ...

REPEATS	TRIGGER	POSITION	TIME1	OUTA1	OUTB1
1	POSA<=POSITION	0	500000	☑	☐
19	Immediate	0	500000	☑	☐
1	Immediate	0	26437500	☐	☑
1	POSA>=POSITION	-20000	500000	☑	☐
19	Immediate	0	500000	☑	☐
1	Immediate	0	26437500	☐	☑
1	POSA<=POSITION	0	500000	☑	☐
19	Immediate	0	500000	☑	☐
1	Immediate	0	26437500	☐	☑
1	POSA>=POSITION	-20000	500000	☑	☐

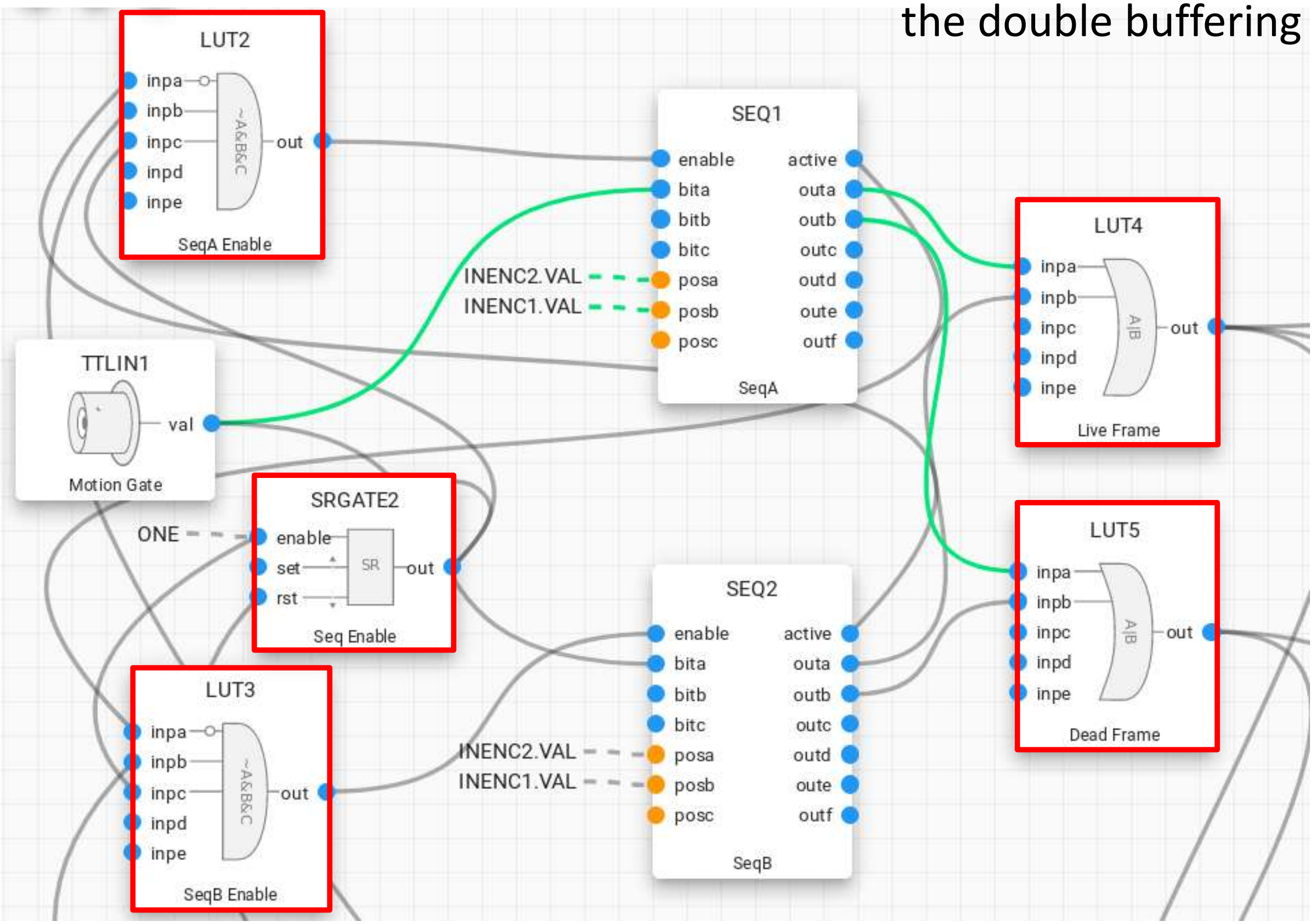
PandA as Master

Second sequence
table used for double buffering
for large trajectories



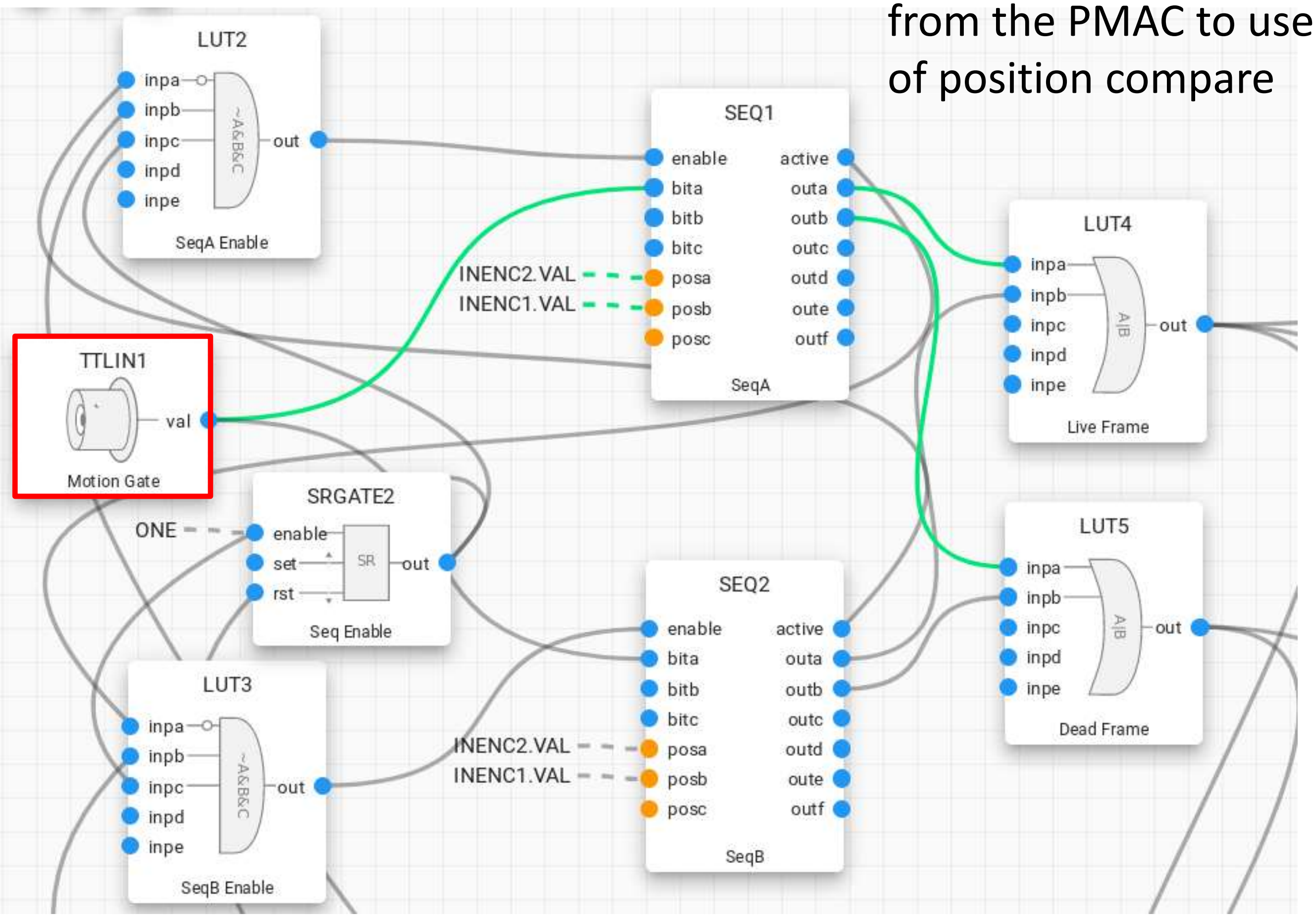
PandA as Master

This is all logic to control the double buffering

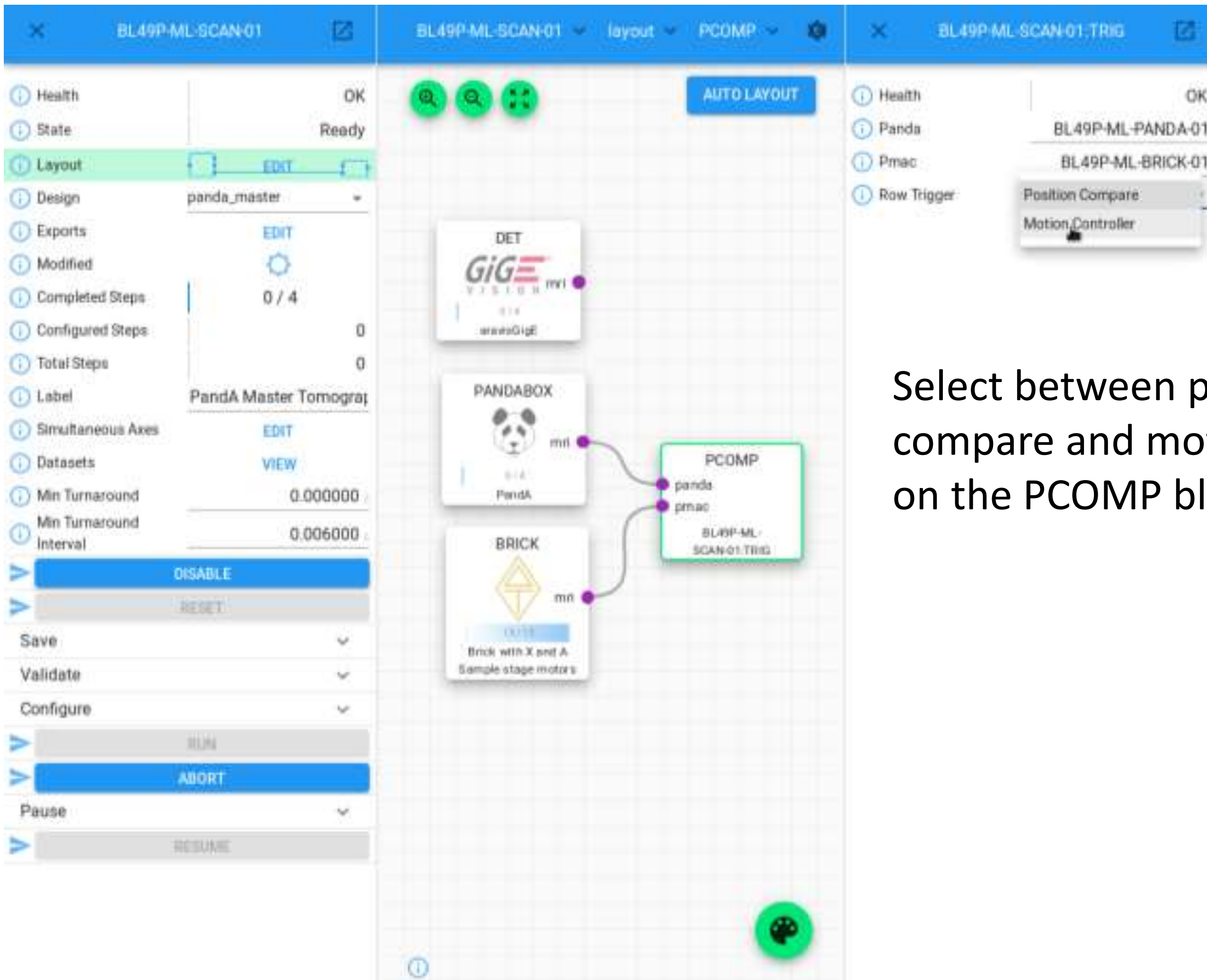


PandA as Master

Takes a motion gate signal from the PMAC to use instead of position compare



Row trigger



Select between position compare and motion gate on the PCOMP block