ACAI: Automated Clock-Oriented Acquisition Infrastructure

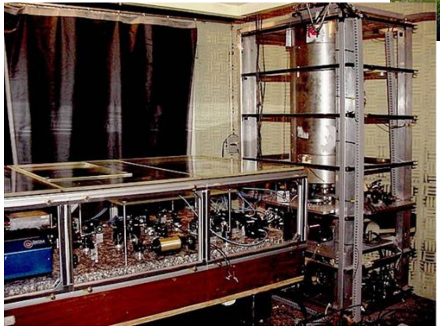
Reusable, Queue-Driven DAQ for Ultra-Precise Clock studies

Sera Conti

Co Supervisors: Federico Alessio, Paolo Durante and Alberto Perro

About Me   Sera Conti

- Computer Engineering (**Universidade de São Paulo**) **+** Computer Science and Engineering (**Politecnico di Milano**)
- Background across robotics, dedicated hardware and electronic instrumentation
- Previously worked on brazilian Atomic Fountain doing **clock stability analyses**
- Currently **LHCb** Computing Group **Summer Student** working on the **TFC Clock Tree**


Context

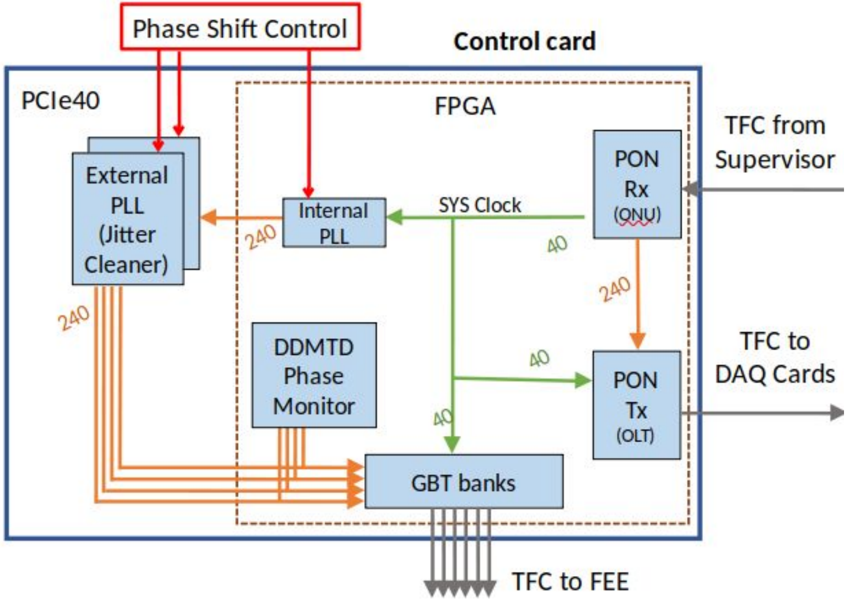


Fig. 1 Current Clock Tree on SOL40 board of TFC

- Timing and Fast Control (**TFC**) System **controls readout** from LHCb **detector**
- **SOL40** controls GBT protocol, **interfacing** Front Enc Electronics (**FEE**) with **TELL40**, responsible for detector readout
- Current **clock tree** provides **insufficient** skew **stability** for the next LHCb **upgrades**

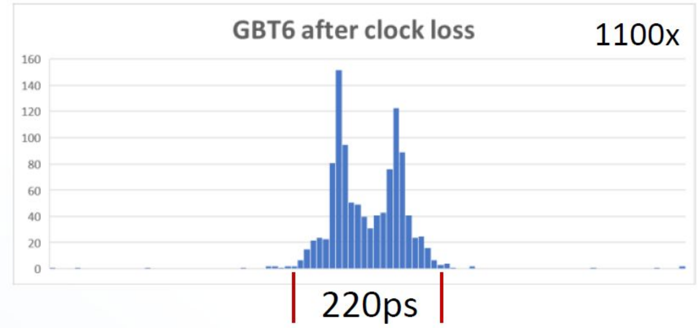


Fig. 2 Previous data on Clock Skew Stability

Challenges

- Current **ad-hoc architecture** and **many proposed alternatives**, but **insufficient data** and not **scalable testing**
- **Search Space Explodes**: clock trees differ on architecture, firmwares, sub detectors, synthesis options...
- **Ad-hoc scripting** is **time consuming** and requires **expertise** to acquire a **single dataset**
- **Hardware Specific** APIs, Scripts, IO Logic, Blocking semantics, timing guarantees…

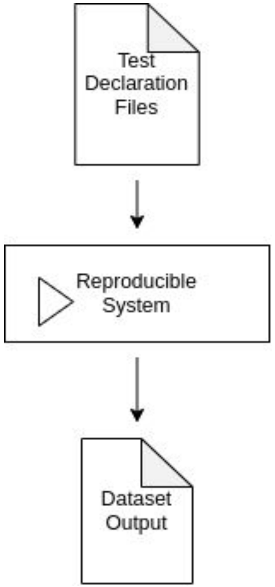
Source: xkcd #292

Requirements and Design Choices

We want a **tool** to make **Lab Testing Fast, Reproducible** and **Usable**

Declarative Testing

- **Reusability** across different testbenches and instruments
- **Decoupling** of tests and the hardware involved
- **Scalability** on testbench size and time scale
- **Performance** on testing ensuring data quality
- **Maintainability** of the code for future improvements
- **Flexibility** for new features

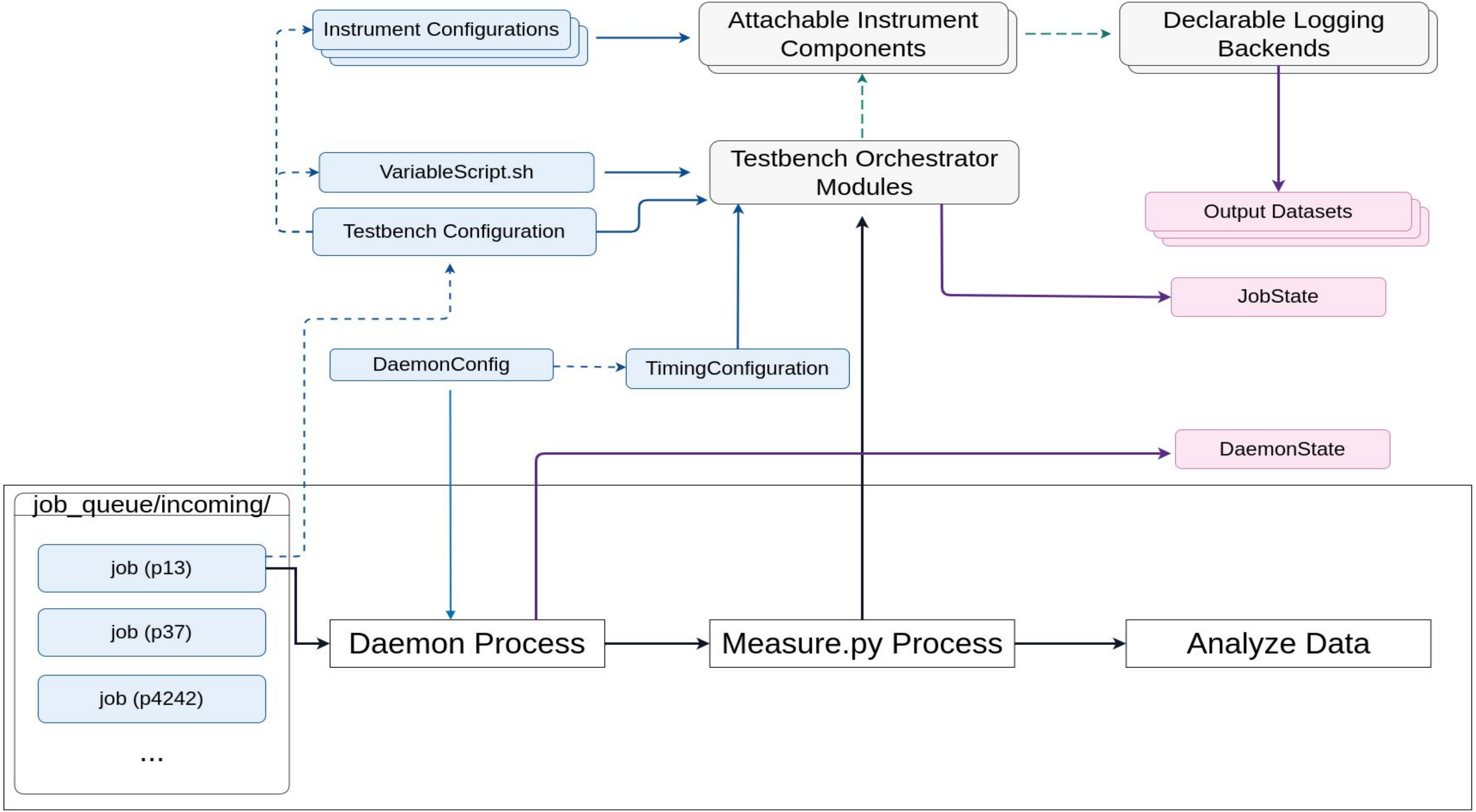


set up

go get a coffee

analyze your precise data ;)

System Overview



Peaking the YAMLs

```yaml
testbench_name: "zdm_long_align"
experiment_bash: variable_script_zdmshift.sh

emulation_mode: false
emulation_data_generator: "realistic"

instruments:

  - instrument_name: 'comp_fpga'
    controller_module: 'src.fpga_meas.fpga_meas_controller'
    controller_class: 'FPGAMeasController'
    config_path: '/opt/clk_meas/config_files/controllers/fpgameas/zdm_long.yaml'
```

testbench_zdm_long.yaml

```yaml
interface_num: 10
use_delay_compensations: false
delays_path: /opt/firmwares/sol40/M/

n_meas: 100
n_exp: 1000
measure_temperature: true

logging:
  backends: [ "csv", "duckdb"]
  buffer_sizes:
    duckdb: 60
    csv: 30

measurements:
  - name: comp_gbt_ref
    clock_type: refclk
    index_expression: 0
    std: true
```

zdm_long.yaml

```yaml
testbench_config: "testbench_zdm_long.yaml"
timing_config: "timing_validation.yaml"
output_path: "/opt/clk_meas/outputs/"
priority: 20
```

job.yaml

```yaml
rtt_mode: freerun
fixed_rtt_meas: 0.01
fixed_rtt_exp: 0.02
force_equal_rtt: false
```

timing_validation.yaml

Achieved Architectural Guarantees and Extra Features

- **Reusability** through **component oriented code** + **configuration YAMLs**
- **Decoupling** through **instrument-specific config** and **controllers**
- **Scalability** through **parameterized size** and **declarative instruments**
- **Performance** separating **core DAQ** and peripheral **modules**
- **Maintainability** of the code for future improvements
- **Flexibility** through **interfaces** and **abstract OO** based implementation

```yaml
emulation_mode: false
emulation_data_generator: "realistic"
```

hardware emulation with configurable data generation for code validation

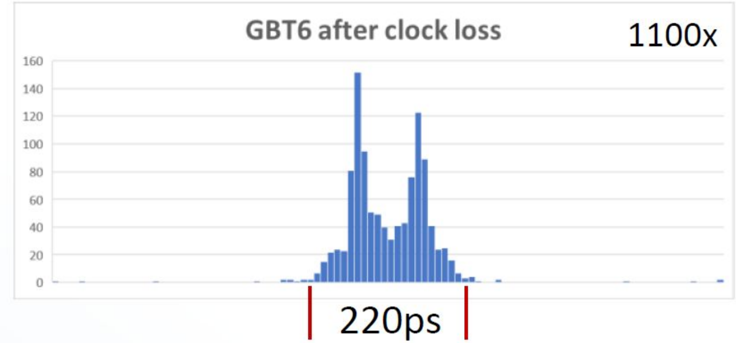
```yaml
rtt_mode: freerun
```

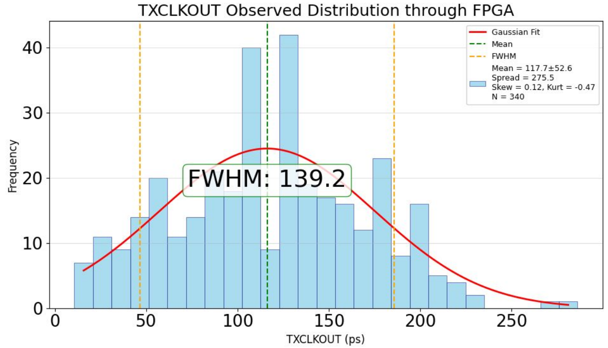
3 timing modes
(performance vs synchronicity)

```yaml
logging:
  backends: [ "csv", "duckdb"]
  buffer_sizes:
    duckdb: 60
    csv: 30
```

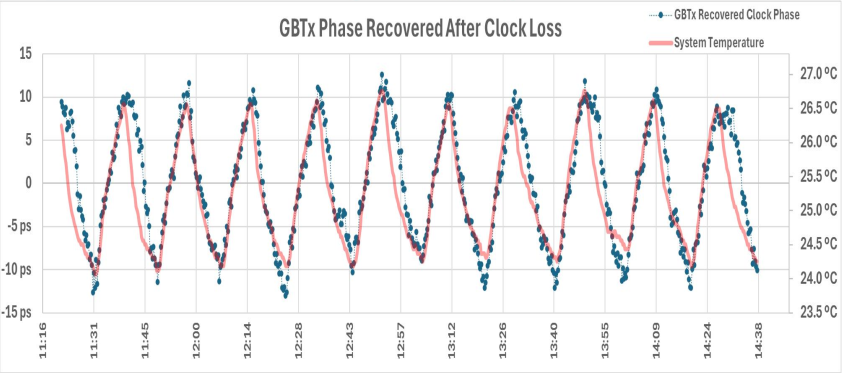
declarable output backends

The Science it Brings - Validation of the Platform

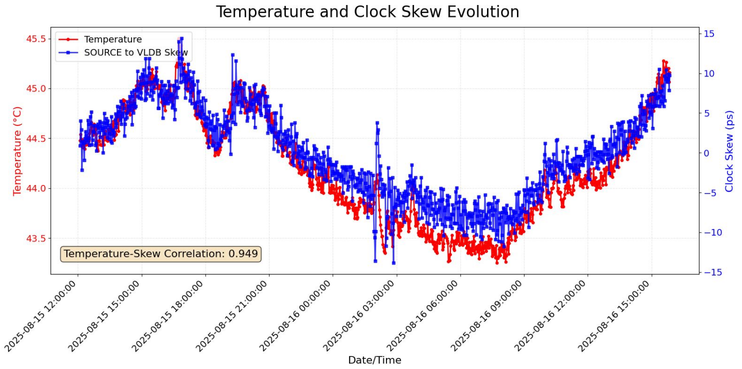




similar 220ps
‘visual spread’



similar
~20ps skew
for
~2.5K drift



The Science it Brings - New Insights

- The **two-peak behaviour** on the skew stability curve was traced back to the **alignment procedure** on the **external PLLs of Marseille** (current) firmware

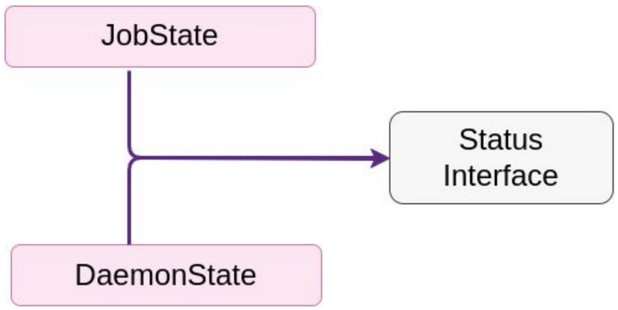
- A new **Zero-Delay Mode Firmware** was tested and seemed to yield **more robust skew results**

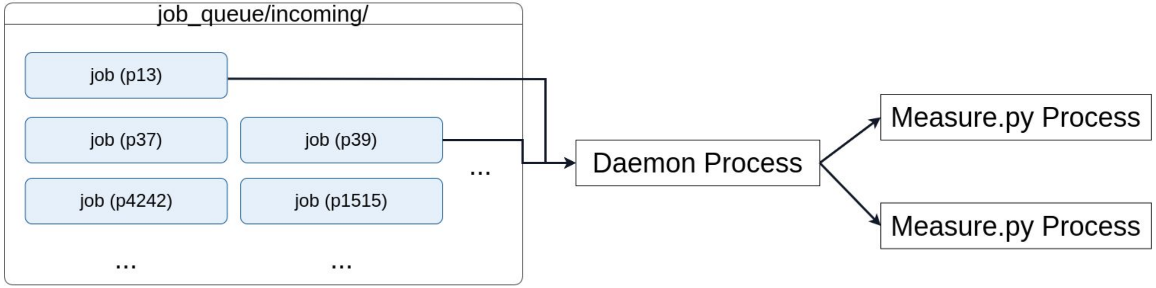
- **Correlations** of the **FPGA** measurements **with oscilloscope** measurements give **promising food for thoughts** about using FPGA readings in **run time** or **automated analysis**

we have: many **useful insights + reusability** to test edge cases **to drive conclusions**

Next Features

- **User Interface** built from state information and reports
- **More Data and StateInfo** parsing daemon terminal output
- **Dataset Versioning** through backend metadata representing the commit + config
- **Test Reconstruction** attaching previous output to emulated data generator
- **Job Level Parallelism**
- **Automated and Runtime Analyses**





obrigade =D

to all my coworkers and colleagues of the CERN Community <3

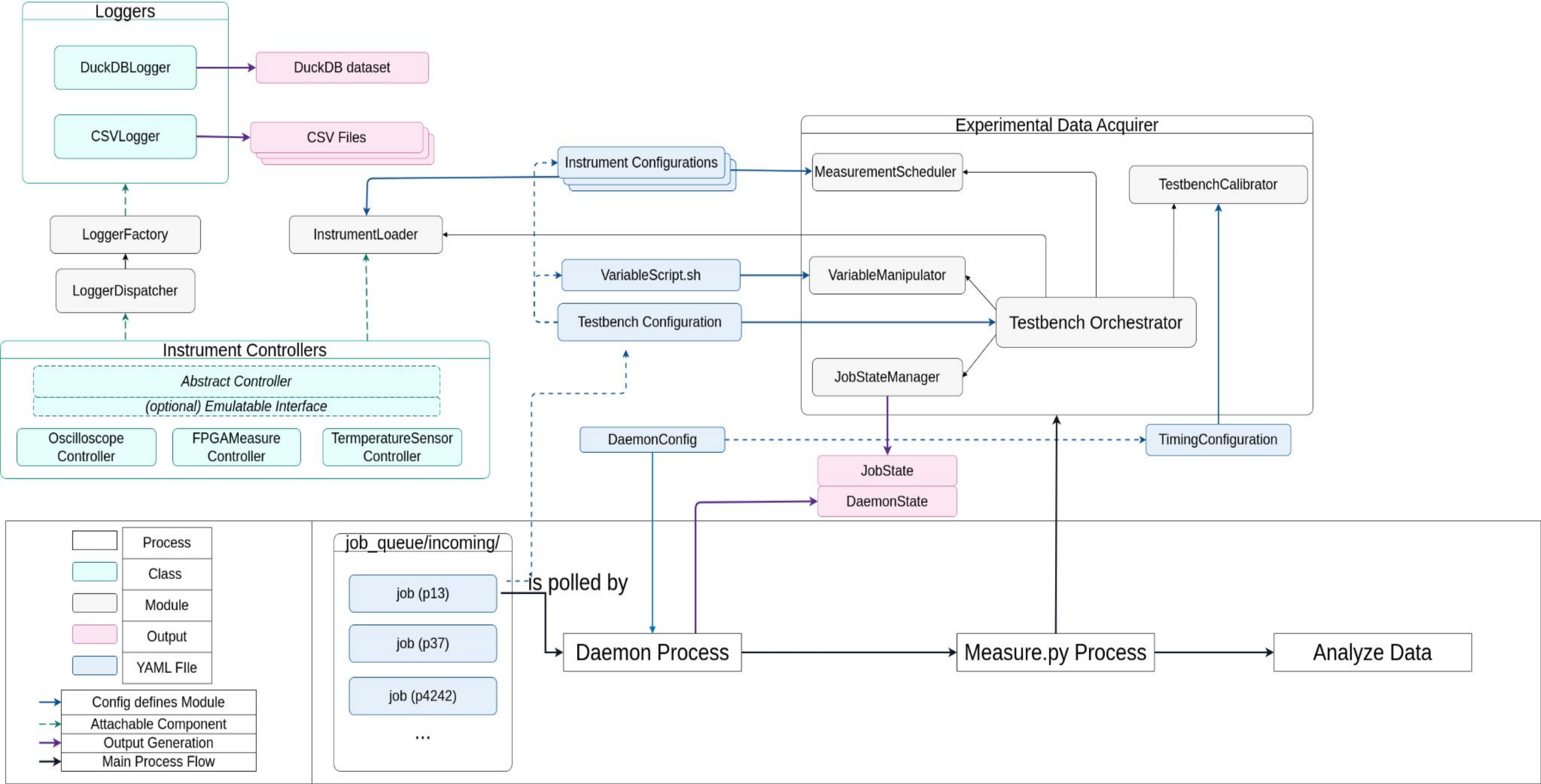
special thanks to my supervisors and for the external help of Maurício Feo

Questions?

gitlab.cern.ch/alperro/clk_studies/-/tree/scope_branch

Extra Slides

Deeper Look on the System



Extra: Writing a new Controller

- **Identify hardware specific API** and divide on the connect() -> setup_device() -> setup_measurement() -> perform_measurment() -> disconnect() pipeline
- **Define config semantics and syntax**
- Abstract controller deals with the logging backends!

```python
@abstractmethod
def setup_device(self):
    pass

@abstractmethod
def setup_measurement(self):
    pass

@abstractmethod
def perform_measurement(self):
    """Perform a measurement and return the results."""
    pass

@abstractmethod
def log_measurement(self, experiment, measurement, should_log=True)
    """Log a measurement if should_log is True."""
    pass

def disconnect(self):
    pass
```

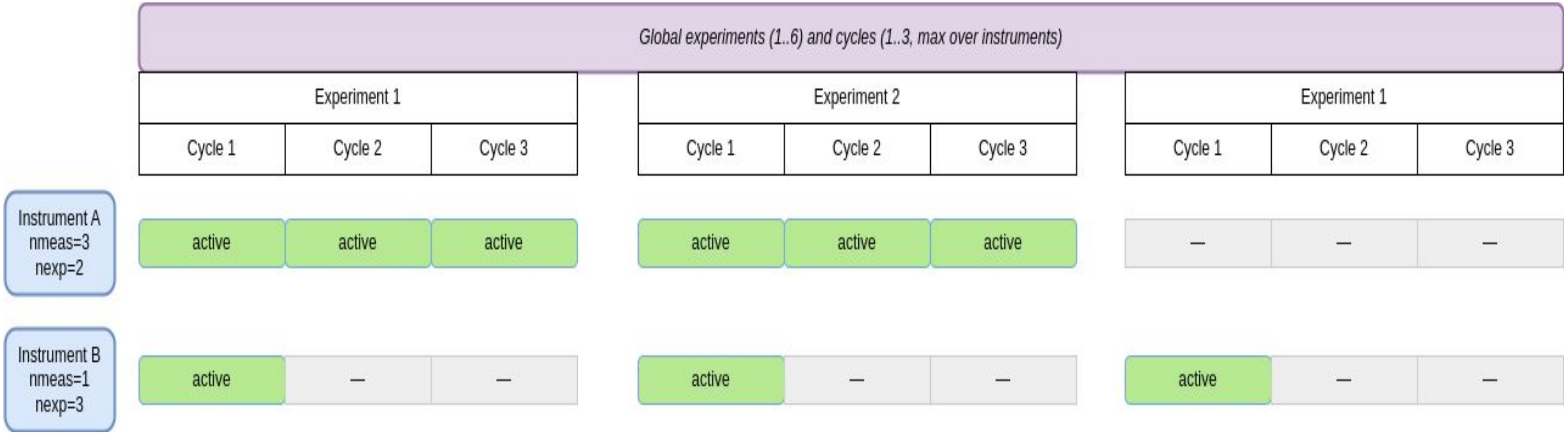
Emulatable Interface allows testing the code without the hardware and/or dependencies

- Define the setup api methods
- Avoid if/elses for maintainability - Lambda Function abstraction serves as function pointer and gives closes behaviour as possible

```python
class EmulatableInterface(ABC):
    """Interface for controllers that support emulation mode."""

    @abstractmethod
    def _setup_real_apis(self):
        """Setup lambda functions for real hardware operations."""
        pass

    @abstractmethod
    def _setup_emulation_apis(self):
        """Setup lambda functions for emulation operations."""
        pass

    @property
    @abstractmethod
    def emulation_mode(self):
        """Return True if controller is in emulation mode."""
        pass
```

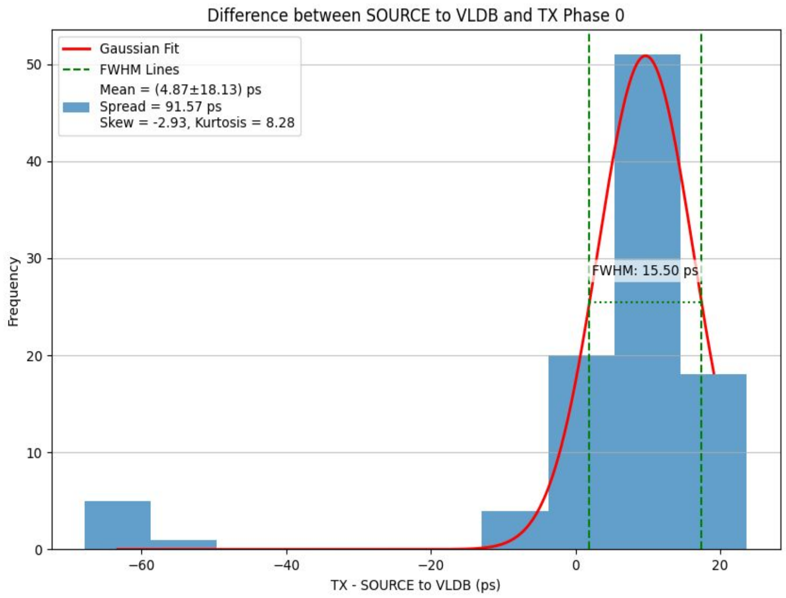
Extra: Synchronicity Modes

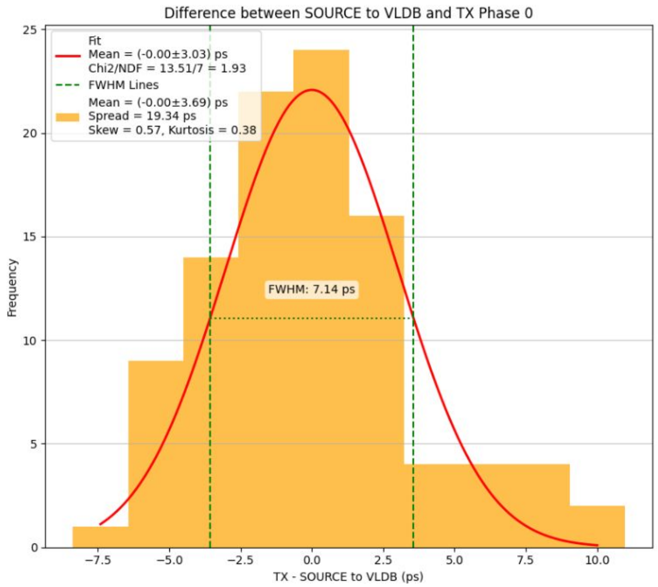
- **Static** predefined by user
- **Dynamic** runs calibration (subset of testbench) to acquire timing metadata
- **Freerun** allows maximum performance and introduces heterogeneous instrument count semantics



Extra: New Insights Explained

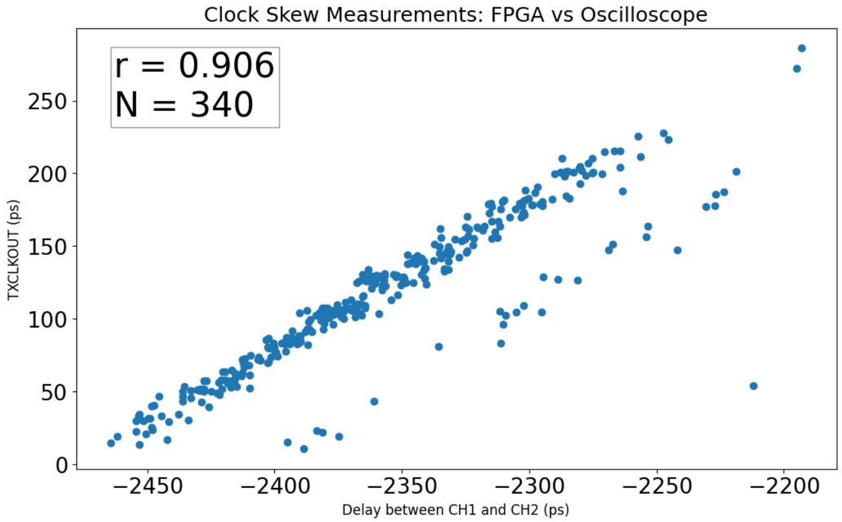
- The **two-peak behaviour** on the skew stability curve was traced back to the **alignment procedure** on the **external PLLs of Marseille** (current) firmware

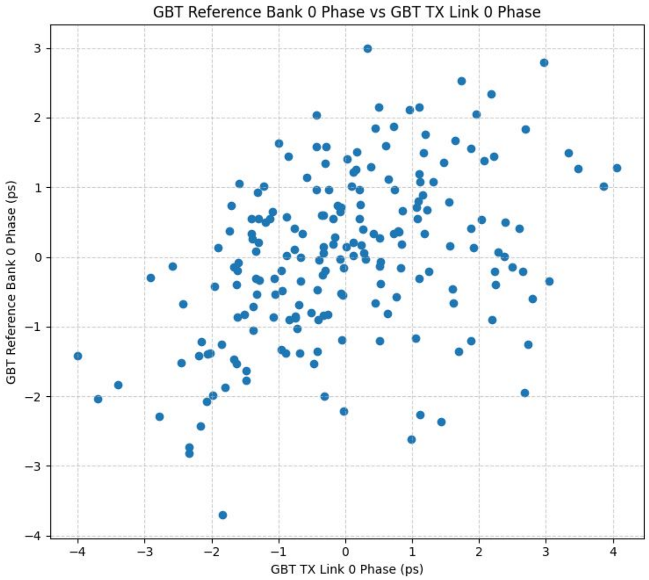




Extra: New Insights Explained

- **Correlations** of the **FPGA** measurements **with oscilloscope** measurements give **promising food for thoughts** about using FPGA readings in **run time** or **automated analysis**





Extra: New Insights Explained

- A new **Zero-Delay Mode Firmware** was tested and seemed to yield **more robust skew results**