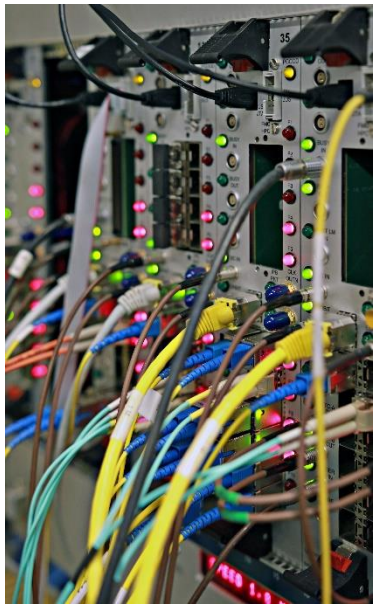
UNIVERSITY OF BIRMINGHAM | SCHOOL OF PHYSICS AND ASTRONOMY

ALICE

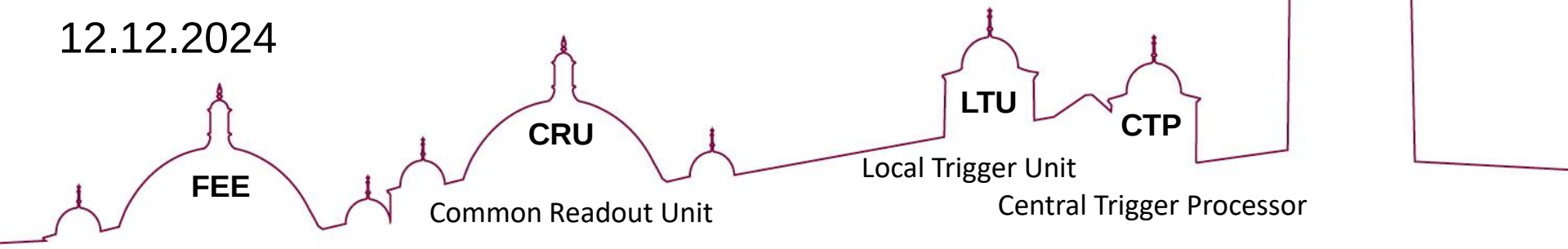
Advanced verification methodology in particle physics



M. Krivda for ALICE CTP

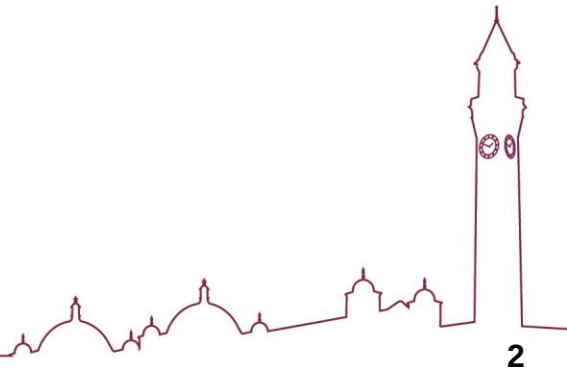
Triggering discoveries in HEP

12.12.2024

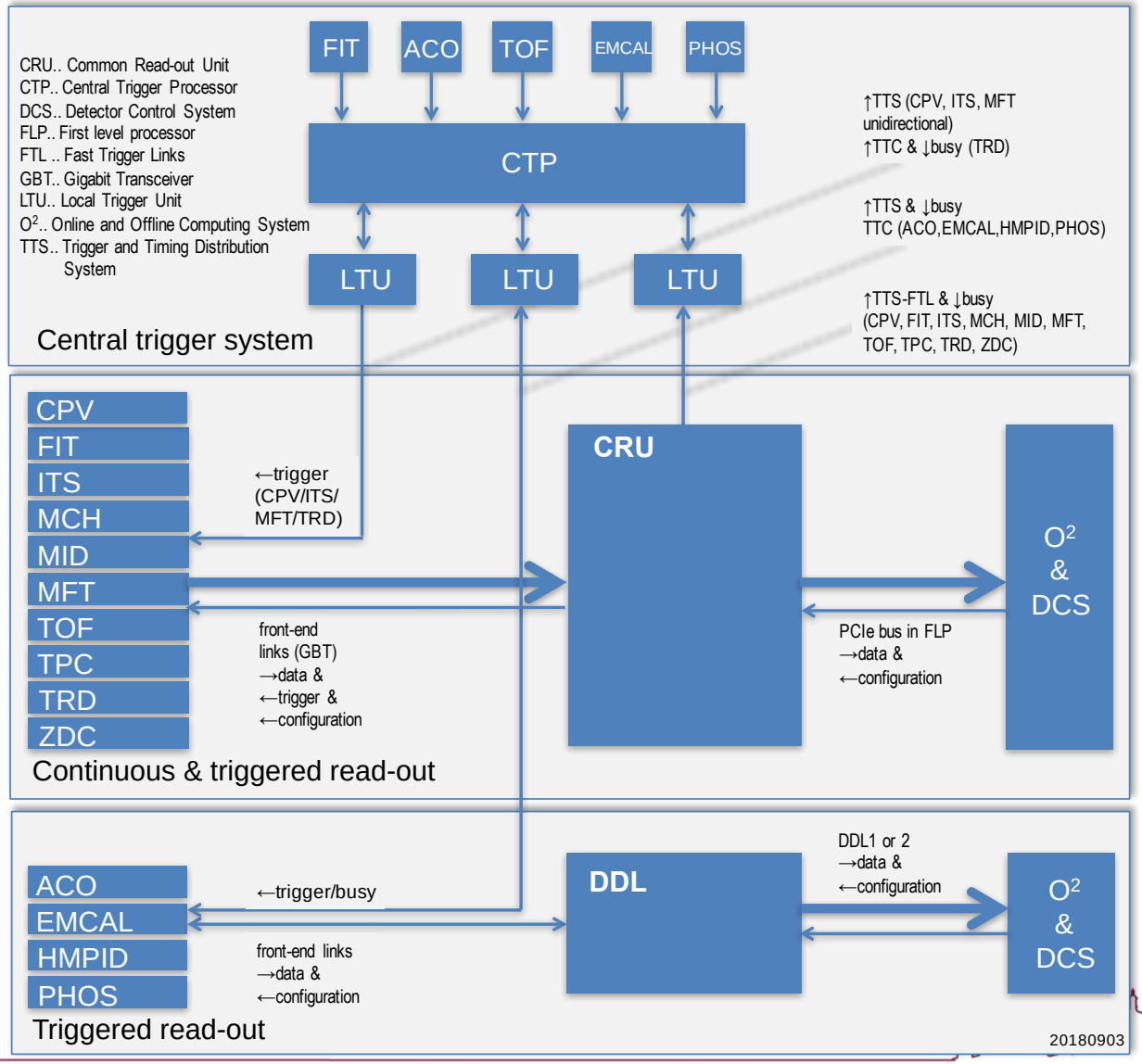


Content

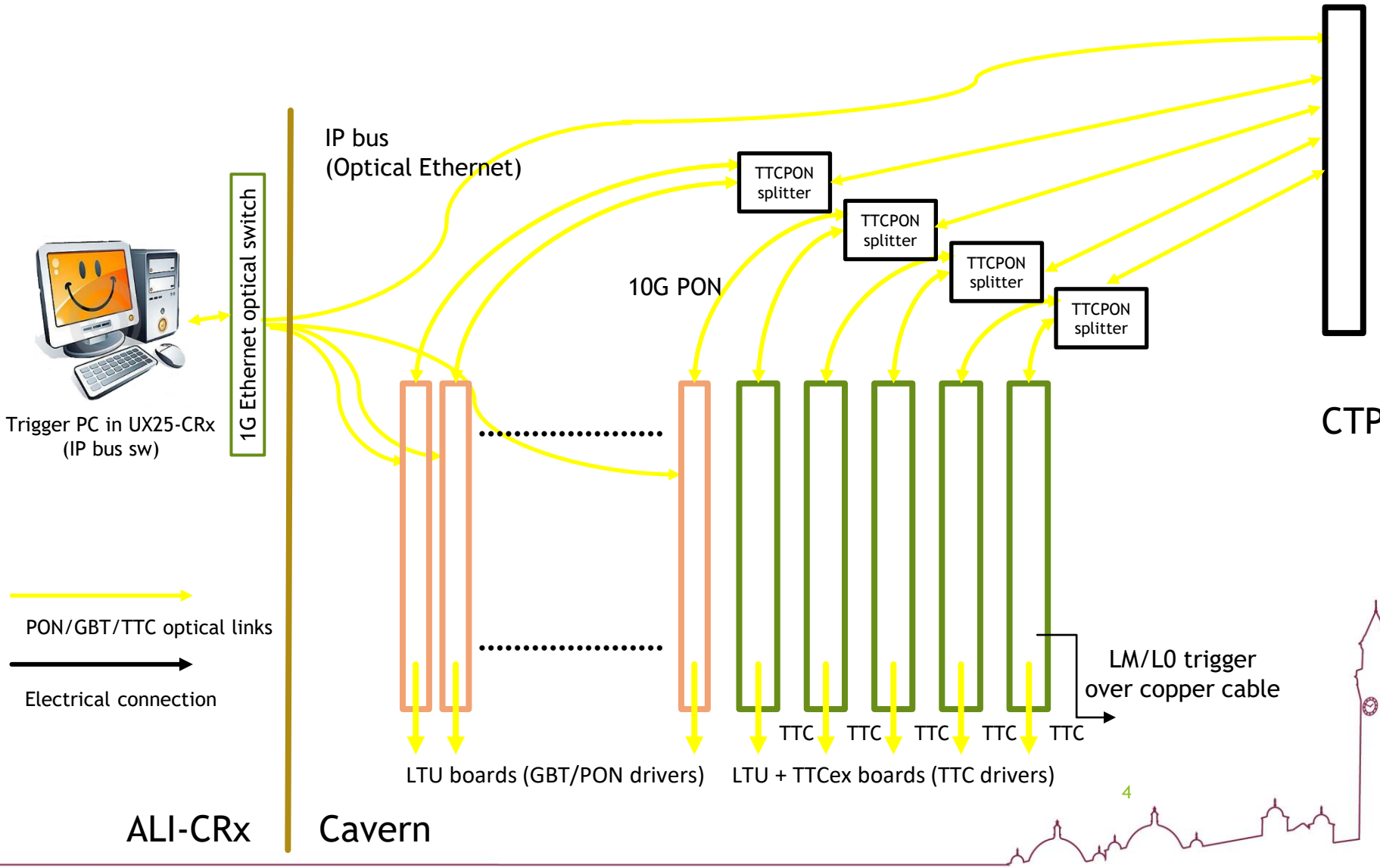
- Alice trigger system overview
- Alice trigger firmware
- The latest firmware upgrades
- Advanced verification methodology
  - Vunit
  - OSVVM
- Summary



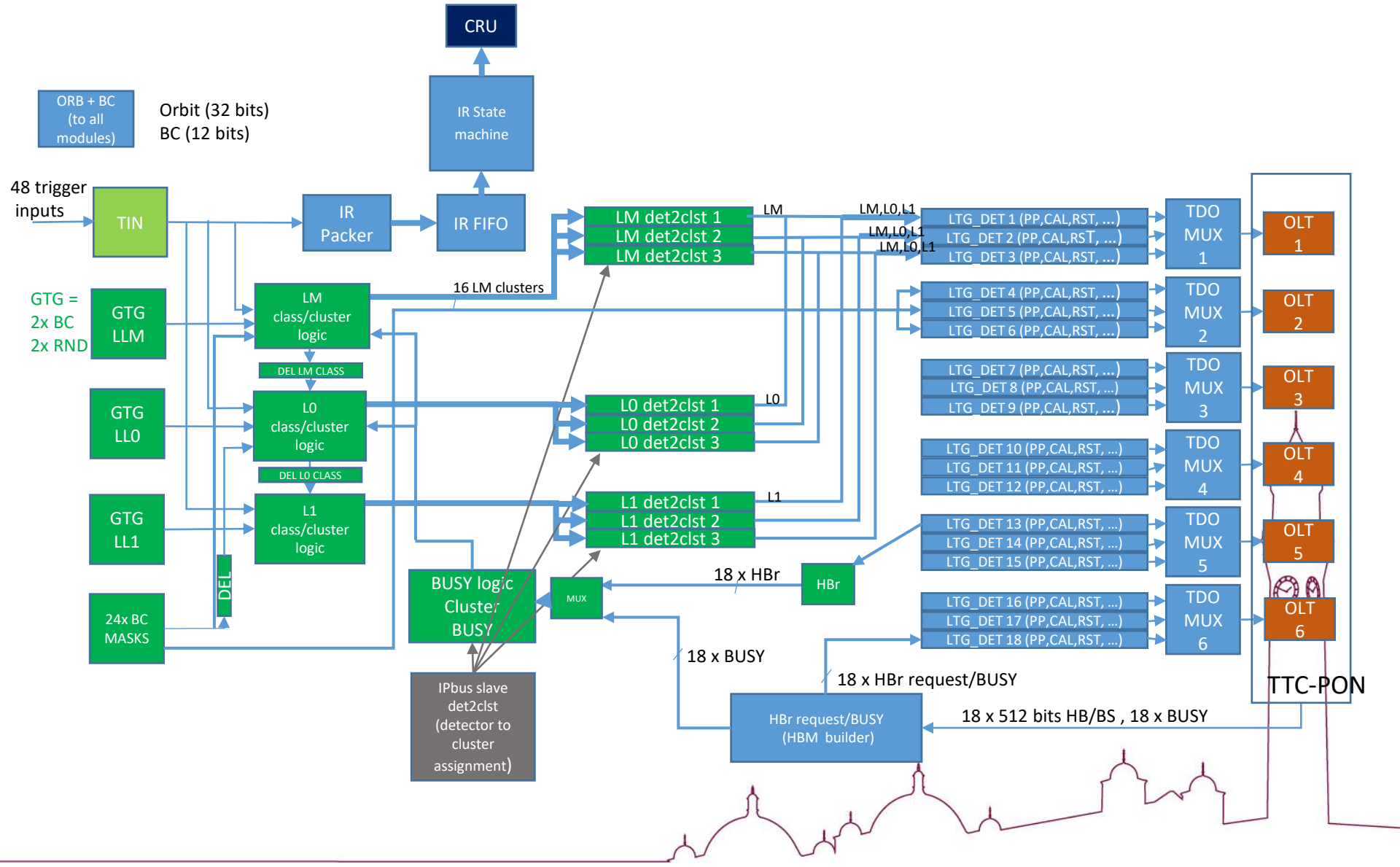
ALICE system block diagram



The trigger-system overview



CTP firmware structure



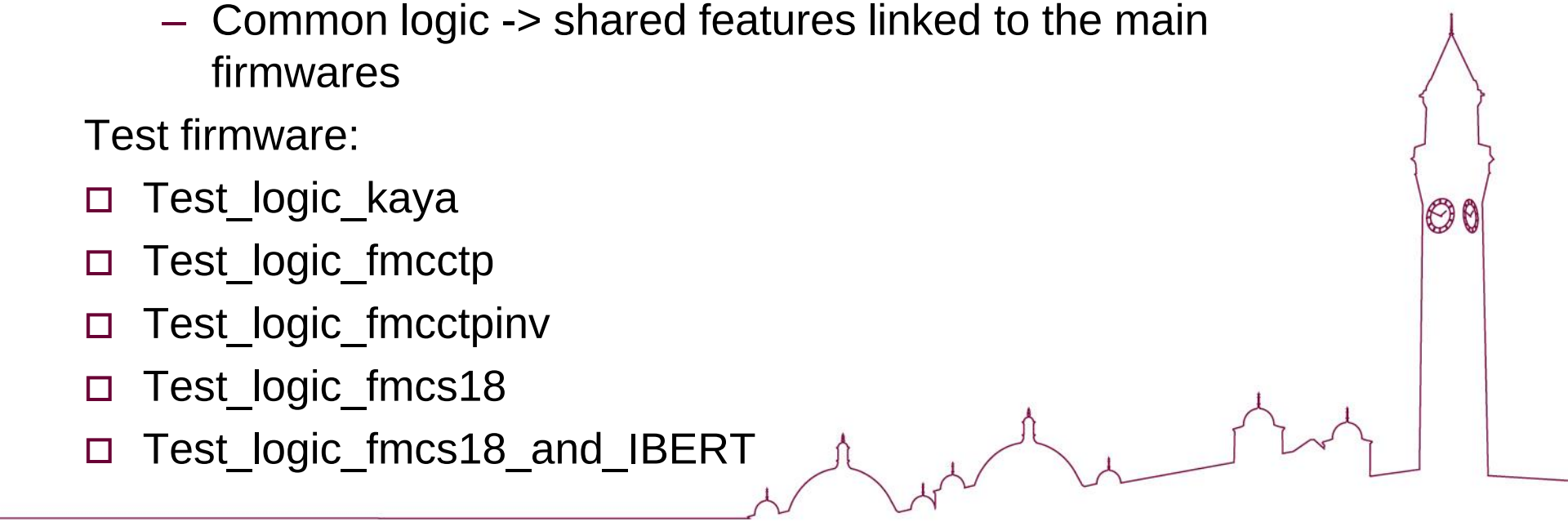
List of firmware types

Production firmware:

- CTP logic
- LTU logic
- LTU_ITSMFT logic
- TICG logic
- TTCit logic
  - Common logic -> shared features linked to the main firmwares

Test firmware:

- Test_logic_kaya
- Test_logic_fmcctp
- Test_logic_fmcctpinv
- Test_logic_fmcs18
- Test_logic_fmcs18_and_IBERT



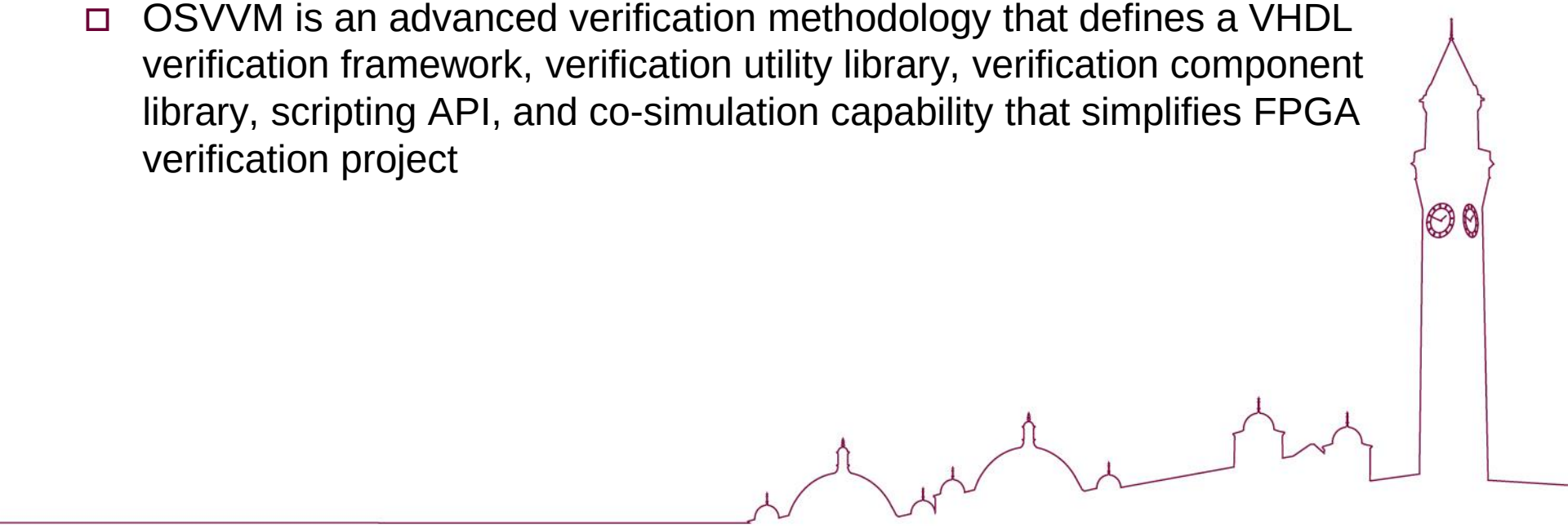
The latest firmware upgrades

- Added IPbus slaves IPROG and ICAP
  Recompiled with VIVADO 2023.2
- Replacement of RARP with DHCP
- Enabled GBT test pattern generator
- IPbus fw 1.13
- Extended clusters from 6 to 8
- Added 3-rd GBT link in place of OLT9

Advanced verification methodology for FPGA design

In order to improve a verification of FPGA design we can use **Vunit** and **OSVVM** (both **open source** tools)

- Vunit is a testing framework for VHDL/SystemVerilog
- OSVVM is an advanced verification methodology that defines a VHDL verification framework, verification utility library, verification component library, scripting API, and co-simulation capability that simplifies FPGA verification project

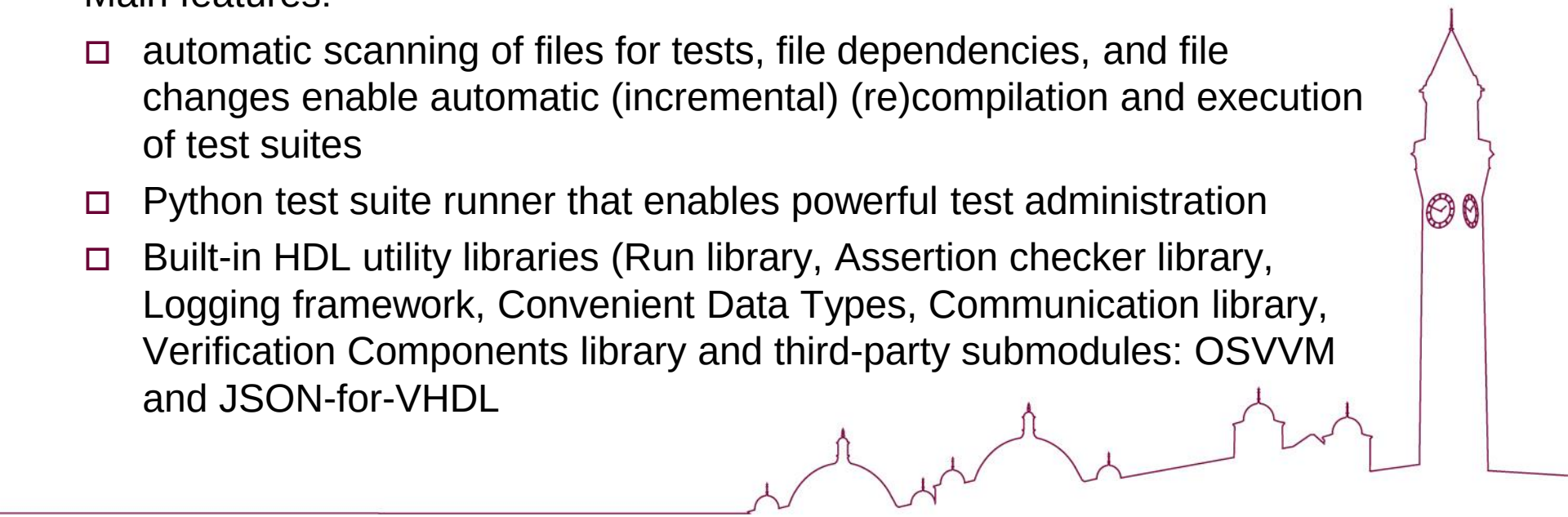


Vunit

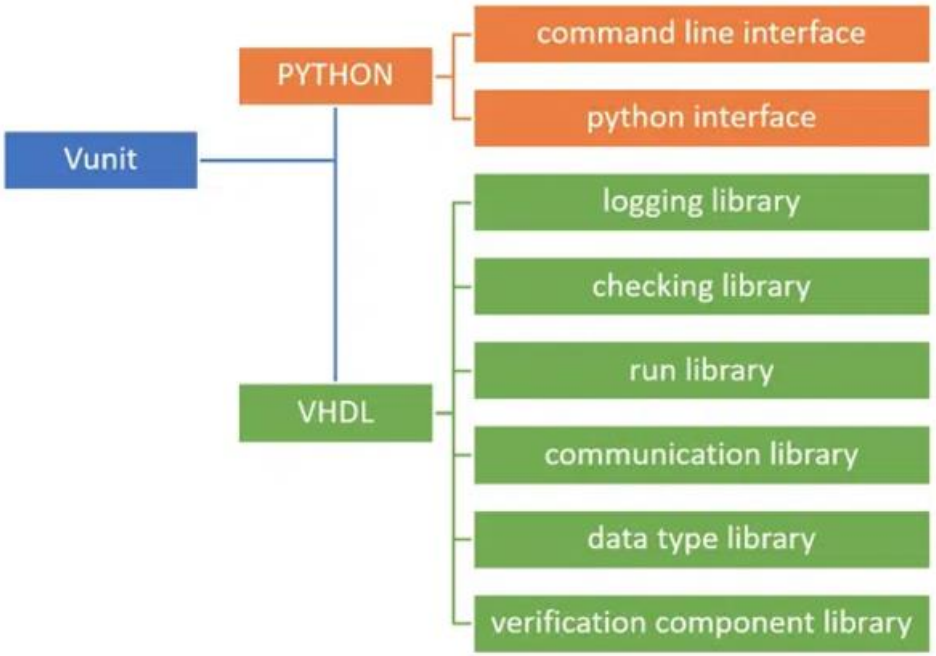
- Reduces the overhead of testing by supporting automatic discovery of test benches and compilation order as well as including libraries for common verification tasks
- Improves the speed of development by supporting incremental compilation and by enabling large test benches to be split up into smaller independent tests
- Supporting Continues Integration (CI)

Main features:

- automatic scanning of files for tests, file dependencies, and file changes enable automatic (incremental) (re)compilation and execution of test suites
- Python test suite runner that enables powerful test administration
- Built-in HDL utility libraries (Run library, Assertion checker library, Logging framework, Convenient Data Types, Communication library, Verification Components library and third-party submodules: OSVVM and JSON-for-VHDL



Vunit



- Vunit commands are available from a simulation tool (in example QuestaSim)
- In example “vunit_restart” is very useful command to restart whole simulation with changes in a source code

Vunit – test case

```
WAIT UNTIL reset <= '0';
WAIT UNTIL falling_edge(clk);
motor_start_in.switch_1 <= '1';  -- turn on switch_1
FOR i IN 0 TO 4 LOOP
  WAIT UNTIL rising_edge(clk);
  WAIT FOR 1 ps;
  check(expr => motor_start_out.red_led = '1',
        msg  => "Expect red_led to be ON '1'");

 check_false(expr => ??motor_start_out.yellow_led,
             msg  => result("for yellow_led when switch_1 on"));

 check_equal(got      => motor_start_out.green_led,
             expected => '0',
             msg      => result("for green_led when switch_1 on"));

  WAIT UNTIL rising_edge(clk);
  WAIT FOR 1 ps;
  check_equal(led_out, STD_LOGIC_VECTOR'("000"),
              result("for led_out when switch_1 on"), warning);
END LOOP;
info("===== TEST CASE FINISHED =====");
```

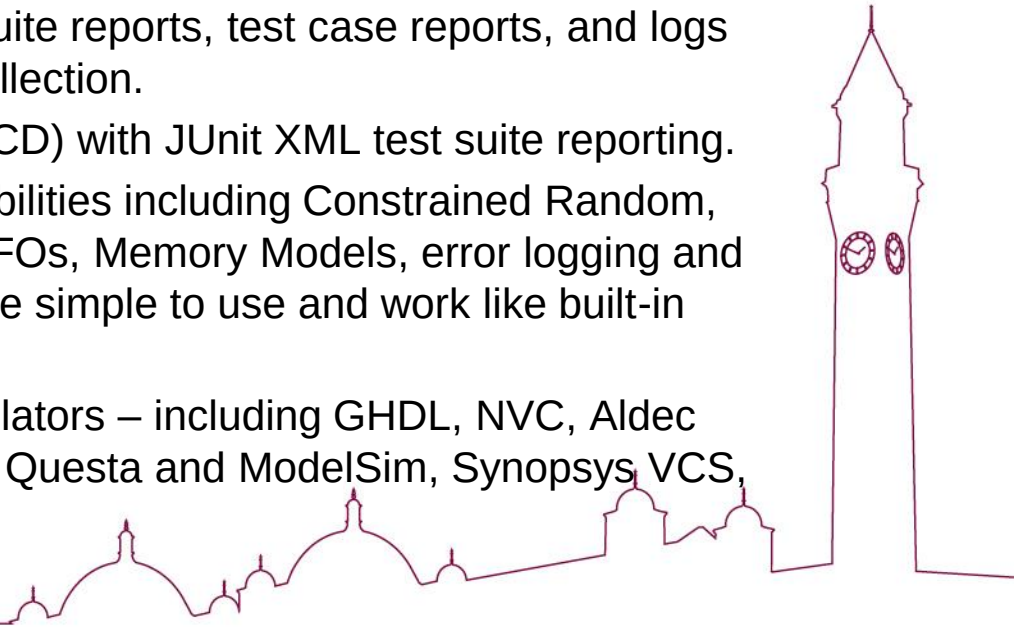
```
Starting tb_lib.motor_start_tb.switch_1_on_output_check
Output file: C:\vunit_tutorial\vunit_out\test_output\tb_lib.motor_start_tb.
switch_1_on_output_check_6df3cd7bf77a9a304e02d3e25d028a92fc541cf5\output.txt
pass (P=1 S=0 F=0 T=1) tb_lib.motor_start_tb.switch_1_on_output_check (1.1 seconds

==== Summary ==================================================
pass tb_lib.motor_start_tb.switch_1_on_output_check (1.1 seconds)
================================================================
pass 1 of 1
================================================================
Total time was 1.1 seconds
Elapsed time was 1.1 seconds
================================================================
All passed!
```

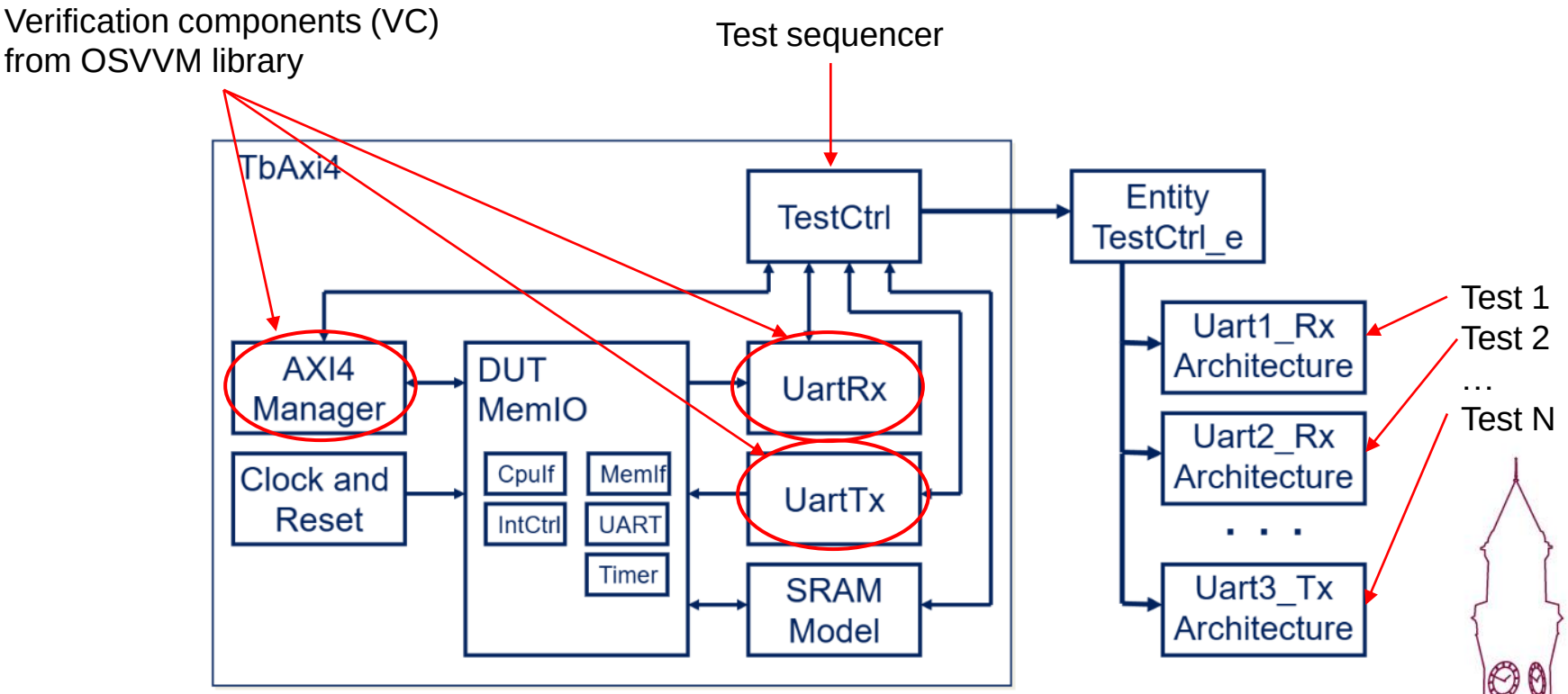
- https://vhdlwhiz.com/getting-started-with-vunit/

OSVVM

- A structured transaction-based framework using verification components that is suitable for all verification tasks
- A Model Independent Transaction (MIT) library that defines a transaction API (procedures such as read, write, send, get, …) and transaction interface (a record) that simplifies writing verification components and test cases.
- Test cases and verification components written in VHDL
- Test cases that are readable and reviewable by the whole team including software and system engineers.
- Test reporting with HTML based test suite reports, test case reports, and logs that facilitate debug and test artifact collection.
- Support for continuous integration (CI/CD) with JUnit XML test suite reporting.
- Powerful and concise verification capabilities including Constrained Random, Functional Coverage, Scoreboards, FIFOs, Memory Models, error logging and reporting, and message filtering that are simple to use and work like built-in language features.
- A common scripting API to run all simulators – including GHDL, NVC, Aldec Riviera-PRO and ActiveHDL, Siemens Questa and ModelSim, Synopsys VCS, and Cadence Xcelium.



OSVVM Testbench Framework



- Connections between the verification components and TestCtrl use VHDL records as an interface
- Connections between the verification components and the DUT are the DUT interfaces (such as UART, AxiStream, AXI4, SPI, and I2C)
- Custom VC written by user

Test

Each test is a separate architecture of TestCtrl

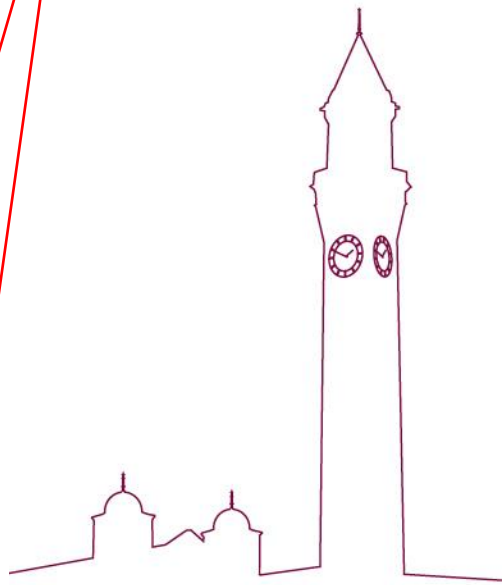
```
architecture UartTx1 of TestCtrl is
  . . .
begin
  ControlProc : process
  begin
    . . .
    WaitForBarrier(TestInit) ;
    . . .
    WaitForBarrier(TestDone, 5 ms) ;
    ReportAlerts ;
    std.env.stop;
  end process ;
  Axi4MProc : process
  begin
    WaitForBarrier(TestInit) ;
    Write(. . .) ;
    WaitForBarrier(DutInit);
    . . .
    WaitForBarrier(TestDone) ;
  end process Axi4MProc ;
  TxProc : process
  begin
    WaitForBarrier(DutInit);
    Send(. . .) ;
    . . .
    WaitForBarrier(TestDone) ;
  end process TxProc ;
  . . .
```

- The TestCtrl architecture consists of a control process

  \+

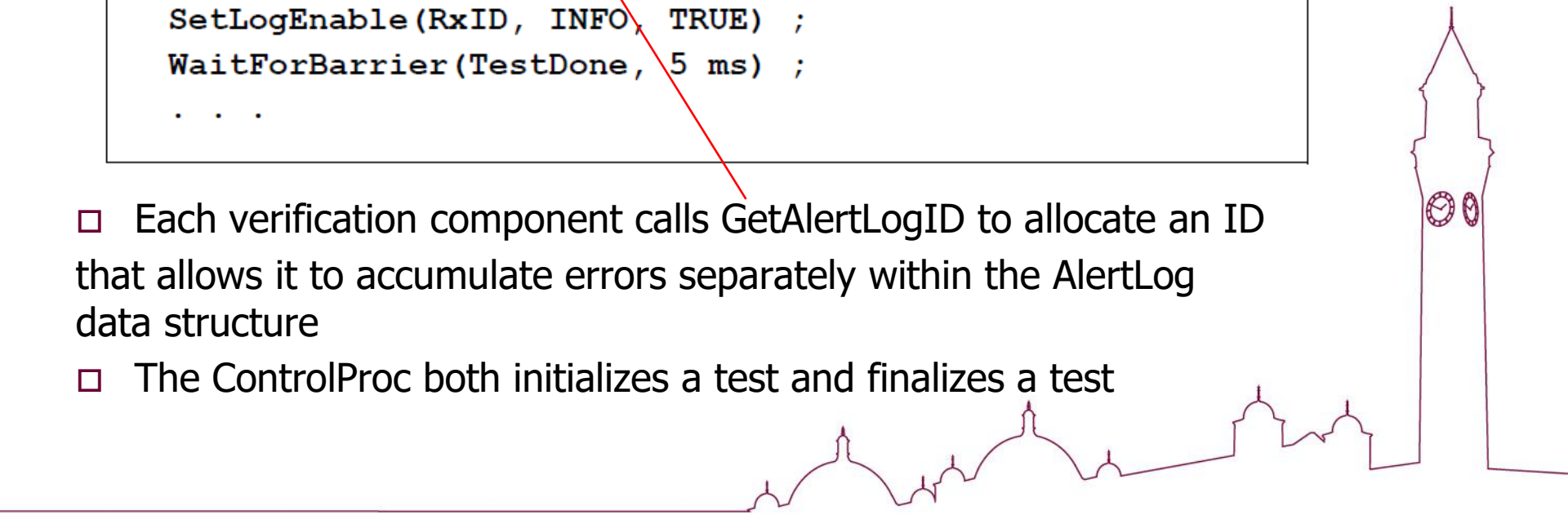
  one process per independent interface

Synchronization of the processes (all processes wait for TestDone signal or Timeout = 5ms)



Test initialization and finalization

```
ControlProc : process
begin
  SetAlertLogName("Test_UartRx_1");
  SB    <= NewID("SB") ;
  TBID <= GetAlertLogID("TB");
  RxID <= GetAlertLogID("UartRx_1");
  SetAlertLogId(SB, "UART_SB") ;
  SetLogEnable(PASSED, TRUE) ;
  SetLogEnable(RxID, INFO, TRUE) ;
  WaitForBarrier(TestDone, 5 ms) ;
  . . .
```



- Each verification component calls GetAlertLogID to allocate an ID that allows it to accumulate errors separately within the AlertLog data structure
- The ControlProc both initializes a test and finalizes a test

A Simple Directed Test

```
TxProc : process

begin
  Send (TRec, X"10") ;


  Send (TRec, X"11") ;
  . . .

end process TxProc
```

```
RxProc : process
  variable RxD : ByteType;
begin
  Get(RRec, RxD) ;
  AffirmIfEqual(TBID, RxD, X"10");

  Check(RRec, X"11") ;
  . . .
end process RxProc ;
```

- The AffirmIfEqual checks its two parameters. It produces a log "PASSED" message if they are equal and alert "ERROR" message otherwise

```
%% Alert ERROR   In TB, Received: 08 /= Expected: 10 at 2150 ns
%% Log   PASSED  In TB, Received: 11 at 3150 ns
```

- The Check transaction checks received value against the supplied expected value. It produces a log "PASSED" message if they are equal and alert "ERROR" message otherwise

```
%% Alert ERROR   In UartRx_1, Received: 8 /= Expected: 10 at 2150 ns
%% Log   PASSED  In UartRx_1, Received: 11 at 3150 ns
```

Model Independent Transactions

- OSVVM improves reusage of interfaces and simplify readability by introducing Model Independent Transactions (MIT)
- Model Independent Transactions observe that many interfaces can be classified as either an address bus interface (AXI, Wishbone, Avalon, X86) or a stream interface (AxiStream, UART, …). For interfaces that fall into one of these categories, OSVVM has defined a set of transactions the interface can support
- The basic set of transactions supported by the different interfaces is shown in a table

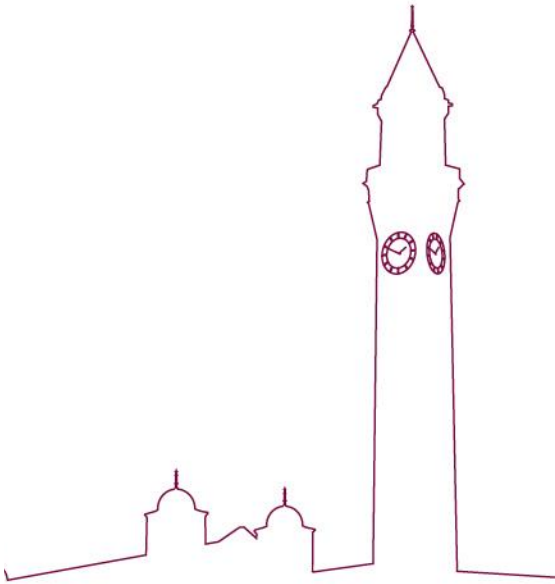
Address Bus Model Independent Transactions

`Write(TransactionRec, X"AAAA", X"DDDD") ;`
`Read (TransactionRec, X"AAAA", DataOut) ;`
`ReadCheck(TransactionRec, X"AAAA", X"DDDD") ;`

Stream Model Independent Transactions

`Send(TransactionRec, X"DDDD") ;`
`Get(Transactionrec, DataOut) ;`
`Check(TransactionRec, X"DDDD") ;`

Common Directive Transactions
`GetTransactionCount(TransactionRec, Count) ;` `SetModelOptions(TransactionRec, Option, OptVal) ;`



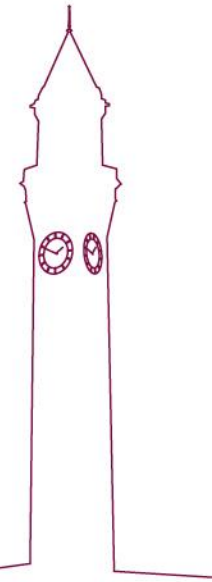
Randomization

- The OSVVM package, RandomPkg, provides a library of randomization utilities
- Example of UART test with normal transactions 70% of the time, parity errors 10% of the time, stop errors 10% of the time, parity and stop errors 5% of the time, and break errors 5% of the time

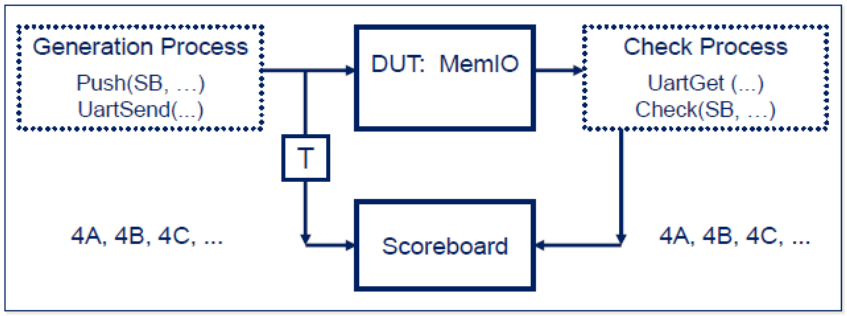
```
TxProc : process
  variable RV : RandomPType ;
  . . .
for I in 1 to 10000 loop
  case RV.DistInt( (70, 10, 10, 5, 5) ) is
    when 0  =>    -- Nominal case   70%
      ErrorMode := UARTTB_NO_ERROR ;
      TxD:= RV.RandSlv(0, 255, Data'length) ;
    when 1  =>    -- Parity Error   10%
      ErrorMode:= UARTTB_PARITY_ERROR ;
      TxD:= RV.RandSlv(0, 255, Data'length) ;
    when  . . .  -- (2, 3, and 4)

  end case ;

  Send(UartTxRec, Data, ErrorMode) ;
end loop ;
```



OSVVM's Scoreboards



- A scoreboard facilitates checking data when there is latency in the system. A scoreboard receives the expected value from the stimulus generation process and checks the value when it is received by the check process

```
use work.ScoreboardPkg_uart.all ;
signal SB : work.Scoreboardpkg_uart.ScoreboardIDType;
. . .
SB <= NewID("SB") ; -- Add to Control Process
```

```
TxProc : process
  variable RV    : RandomPType;
  . . .
begin
  for I in 1 to 10000 loop
    case RV.DistInt((. . .)) is
      . . .
    end case ;
    Push(SB, (TxD, ErrorMode));
    Send(TRec, TxD, ErrorMode);
  end loop ;
  . . .
```

```
RxProc : process
  variable RxD   : ByteType;
begin

  for I in 1 to 10000 loop

    Get(RRec, RxD, RxErrorMode);
    Check(SB, (RxD, RxErrorMode));
  end loop ;
  . . .
```

Received

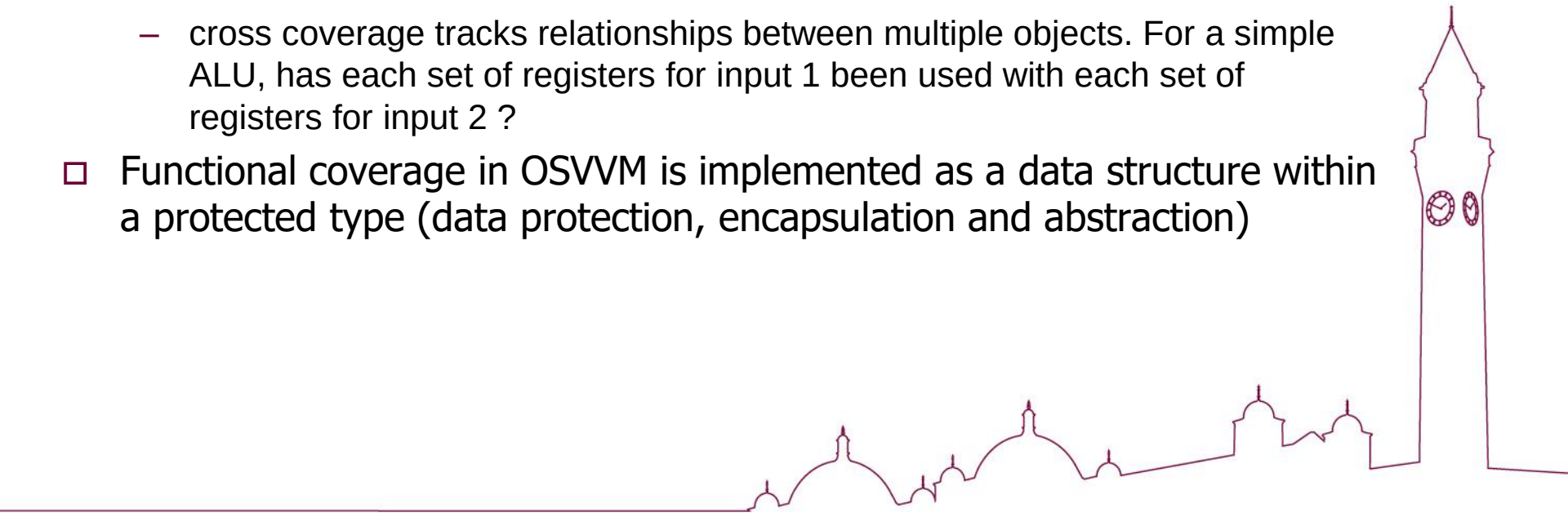
Checked in the scoreboard

Pushed to the scoreboard

Transmitted

Functional Coverage

- If a test uses constrained random, functional coverage is needed to determine if the test did something useful
- A functional coverage is recommended to assure that a directed test actually did everything that was intended
- There are two categories of functional coverage: coverage and cross coverage
  - coverage tracks relationships within a single object. For a UART, were transfers with no errors, parity errors, stop bit errors, parity and stop bit errors, and break errors seen?
  - cross coverage tracks relationships between multiple objects. For a simple ALU, has each set of registers for input 1 been used with each set of registers for input 2 ?
- Functional coverage in OSVVM is implemented as a data structure within a protected type (data protection, encapsulation and abstraction)



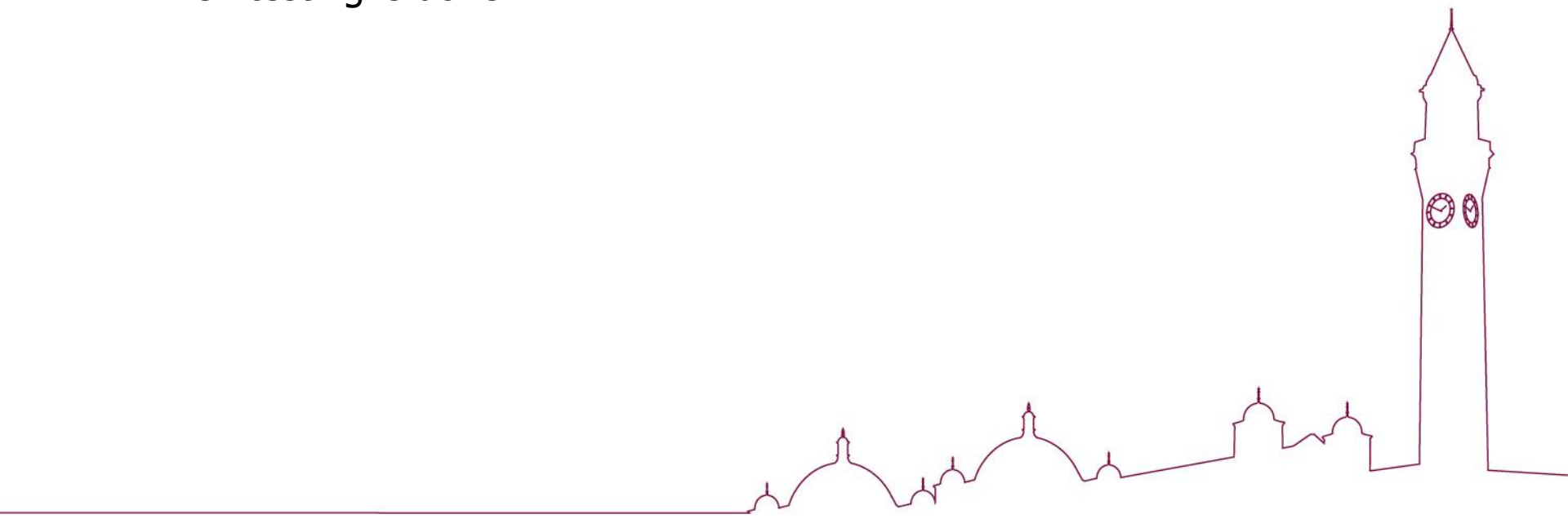
Functional Coverage

```
architecture CR_1 of TestCtrl is
  signal RxCov : CoverageIDType ;  -- define coverage object
  . . .
begin
. . .
RxProc : process
  . . .
begin
  RxCov <= NewID("RxCov", TbID) ;
  wait for 0 ns ;
  -- Define coverage model
  AddBins(RxCov, GenBin(1) ) ;          -- Normal
  AddBins(RxCov, GenBin(3) ) ;          -- Parity Error
  AddBins(RxCov, GenBin(5) ) ;          -- Stop Error
  AddBins(RxCov, GenBin(7) ) ;          -- Parity + Stop
  AddBins(RxCov, GenBin(9, 15, 1) ) ; -- Break
  for I in 1 to 10000 loop
    Get(RRec, RxD, RxErrorMode);
    Check(SB, (RxD, RxErrorMode));
    ICover(RxCov, RxErrorMode) ;       -- Collect functional coverage
  end loop ;
  . . .
  WriteBin(RxCov) ;                    -- Print coverage results
```

- "AddBins" is called to construct the functional coverage model
- "Get" is used to retrieve stimulus
- "Icover" is called to record the coverage
- "WriteBin" prints the coverage results.

Code coverage and Functional coverage

- A code coverage (feature available directly in simulation tools) only tracks a code execution. The code coverage cannot track all OSVVM features since the information is not in the code.
- On the other hand, if a design's code coverage does not reach 100% then there are untested items and testing is not done.
- Both, code coverage and functional coverage are needed to determine when testing is done.



OSVVM alerts

- OSVVM alerts are used to check for invalid conditions on an interface or library subprogram
- Alerts report and count errors
- Alerts have the levels FAILURE, ERROR and WARNING.
  - FAILURE level alerts cause a simulation to stop
  - ERROR and WARNING do not cause a simulation to stop

Check if iCE is together with iWE and iOE

```
 SimultaneousAccessCheck: process
begin
  wait on iCE, iWE, iOE ;
  AlertIf(SramAlertID, (iCE and iWE and iOE) = '1',
      "nCE, nWE, and nOE are all active") ;
end process SimultaneousAccessCheck ;
```

- When a test completes, all errors reported by Alert (and AffirmIf) can be reported using ReportAlerts
- Alerts can be enabled or disabled via SetAlertEnable
- The stopping behaviour of Alert levels can be changed with SetAlertStopCount

```
-- Turn off Warnings for a verification component
SetAlertEnable(UartRxAlertLogID, WARNING, FALSE) ;

-- If get 20 ERRORs stop the test
SetAlertStopCount(ERROR, 20) ;
```

Test Reporting

- ReportAlerts to generate a test completion report with PASSED/FAILED
- WriteAlertSummaryYaml to add an alert summary to the build report
- WriteAlertYaml to generate a detailed tests alert report
- WriteCovYaml to generate a detailed coverage report

Alert Reports (text based) when using ID based reporting example:

```
%% DONE  PASSED  TbUart_SendGet1  Passed: 48  Affirmations Checked: 48  at 100000 ns
```

```
%% DONE  FAILED  TbUart_SendGet1  Total Error(s) = 7  Failures: 0  Errors: 7  Warnings: 0
Passed: 41  Affirmations Checked: 48  at 100000 ns
%%    Default                     Failures: 0  Errors: 0  Warnings: 0  Passed: 0
%%    OSVVM                       Failures: 0  Errors: 0  Warnings: 0  Passed: 0
%%    TB                          Failures: 0  Errors: 0  Warnings: 0  Passed: 0
%%    AxiM_1                      Failures: 0  Errors: 0  Warnings: 0  Passed: 0
%%    UartTx_1                    Failures: 0  Errors: 0  Warnings: 0  Passed: 0
%%    UartRx_1                    Failures: 0  Errors: 7  Warnings: 0  Passed: 0
%%      UartRx_1: Data Check      Failures: 0  Errors: 7  Warnings: 0  Passed: 41
```

Test Reporting

Alert Reports - YAML and HTML example:

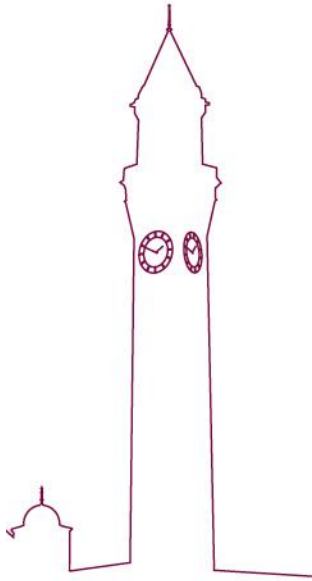
TbAxi4_RandomReadWrite Alert Report

▾ TbAxi4_RandomReadWrite Alert Settings

Setting		Value	Description
FailOnWarning		true	If true, warnings are a test error
FailOnDisabledErrors		true	If true, Disabled Alert Counts are a test error
FailOnRequirementErrors		true	If true, Requirements Errors are a test error
External	Failures	0	Added to Alert Counts in determine total errors
	Errors	0	
	Warnings	0	
Expected	Failures	0	Subtracted from Alert Counts in determine total errors
	Errors	0	
	Warnings	0	

▾ TbAxi4_RandomReadWrite Alert Results

Name	Status	Checks		Total Errors	Alert Counts			Requirements		Disabled Alert Counts		
		Passed	Total		Failures	Errors	Warnings	Passed	Checked	Failures	Errors	Warnings
TbAxi4_RandomReadWrite	PASSED	3000	3000	0	0	0	0	0	0	0	0	0
Default	PASSED	1750	1750	0	0	0	0	0	0	0	0	0
OSVVM	PASSED	0	0	0	0	0	0	0	0	0	0	0
subordinate_1	PASSED	0	0	0	0	0	0	0	0	0	0	0
subordinate_1: Protocol Error	PASSED	0	0	0	0	0	0	0	0	0	0	0
subordinate_1: Data Check	PASSED	0	0	0	0	0	0	0	0	0	0	0
subordinate_1: No response	PASSED	0	0	0	0	0	0	0	0	0	0	0
manager_1	PASSED	0	0	0	0	0	0	0	0	0	0	0
manager_1: Protocol Error	PASSED	0	0	0	0	0	0	0	0	0	0	0
manager_1: Data Check	PASSED	250	250	0	0	0	0	0	0	0	0	0
manager_1: No response	PASSED	0	0	0	0	0	0	0	0	0	0	0
manager_1: WriteResponse Scoreboard	PASSED	500	500	0	0	0	0	0	0	0	0	0
manager_1: ReadResponse Scoreboard	PASSED	500	500	0	0	0	0	0	0	0	0	0
manager_1: WriteBurstFifo	PASSED	0	0	0	0	0	0	0	0	0	0	0
manager_1: ReadBurstFifo	PASSED	0	0	0	0	0	0	0	0	0	0	0



Test Reporting

Coverage Reports –
YAML and HTML example:

Uart7_Random_part3 Coverage Report

Total Coverage: 100.00

▼ UART_RX_STIM_COV Coverage Model **Coverage: 100.0**

▼ UART_RX_STIM_COV Coverage Settings

CovWeight	1
Goal	100.0
WeightMode	at_least
Seeds	824213985 792842968
CountMode	count_first
IllegalMode	illegal_on
Threashold	45.0
ThresholdEnable	0
TotalCovCount	100
TotalCovGoal	100

▼ UART_RX_STIM_COV Coverage Bins

Name	Type	Mode	Data	Idle	Count	AtLeast	Percent Coverage
NORMAL	Count	1 to 1	0 to 255	0 to 0	63	63	100.0
NORMAL	Count	1 to 1	0 to 255	1 to 15	7	7	100.0
PARITY	Count	3 to 3	0 to 255	2 to 15	11	11	100.0
STOP	Count	5 to 5	1 to 255	2 to 15	11	11	100.0
PARITY_STOP	Count	7 to 7	1 to 255	2 to 15	6	6	100.0
BREAK	Count	9 to 15	11 to 30	2 to 15	2	2	100.0
Total Percent Coverage:	100.0						

▼ UART_RX_COV Coverage Model **Coverage: 100.0**

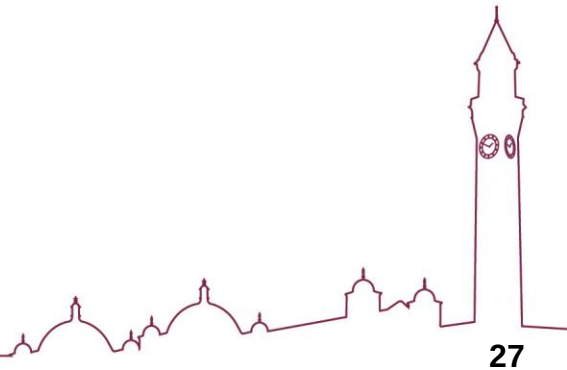
▶ UART_RX_COV Coverage Settings

▼ UART_RX_COV Coverage Bins

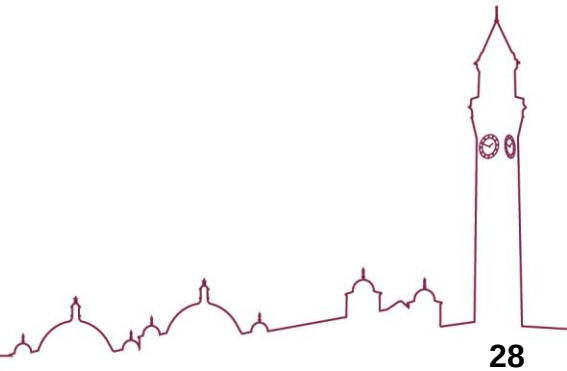
Name	Type	Mode	Count	AtLeast	Percent Coverage
NORMAL	Count	1 to 1	70	1	7000.0
PARITY	Count	3 to 3	11	1	1100.0

Summary

- Vunit + OSVVM is very effective tool for a firmware verification
- OSVVM ver. 2024.09 has many advanced features for AXI4 bus
- CI should be added to continuously verify future firmware changes
- An effort to convert Alice trigger firmware to Vunit+OSVVM has started



Back-up slides



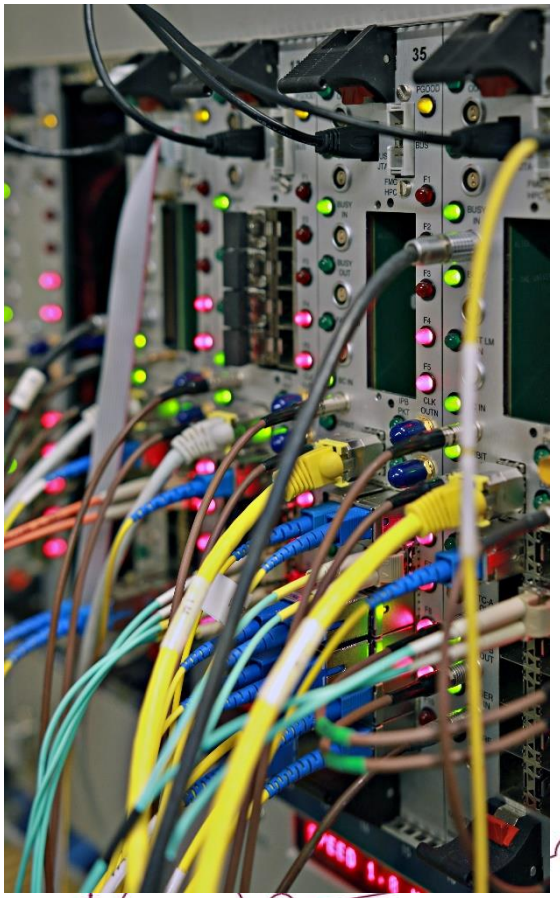
ALICE Trigger system

- 14 detectors (9 with TTC-PON system, 4 with TTC system, TRD (TTC+PON))
- 4 Triggering detectors (FIT, EMC, PHO, TOF; 34 trigger inputs)
- Trigger Input latencies (time from interaction to signal input at CTP)
  - 425 ns (contributing detector – FIT) → Interaction (Minimum Bias) trigger
  - 1.2 μs (contributing det. – EMC, PHO, TOF)
  - 6.1 μs (EMC)
- Each detector sees only ONE trigger (LM, L0 or L1)
  - Except special cases (PHOS and HMPID)
- Electronics must survive in magnetic field of ~11 mT (rack location near/below the dipole magnet)
- Random jitter on the clock ~10 ps at FEE

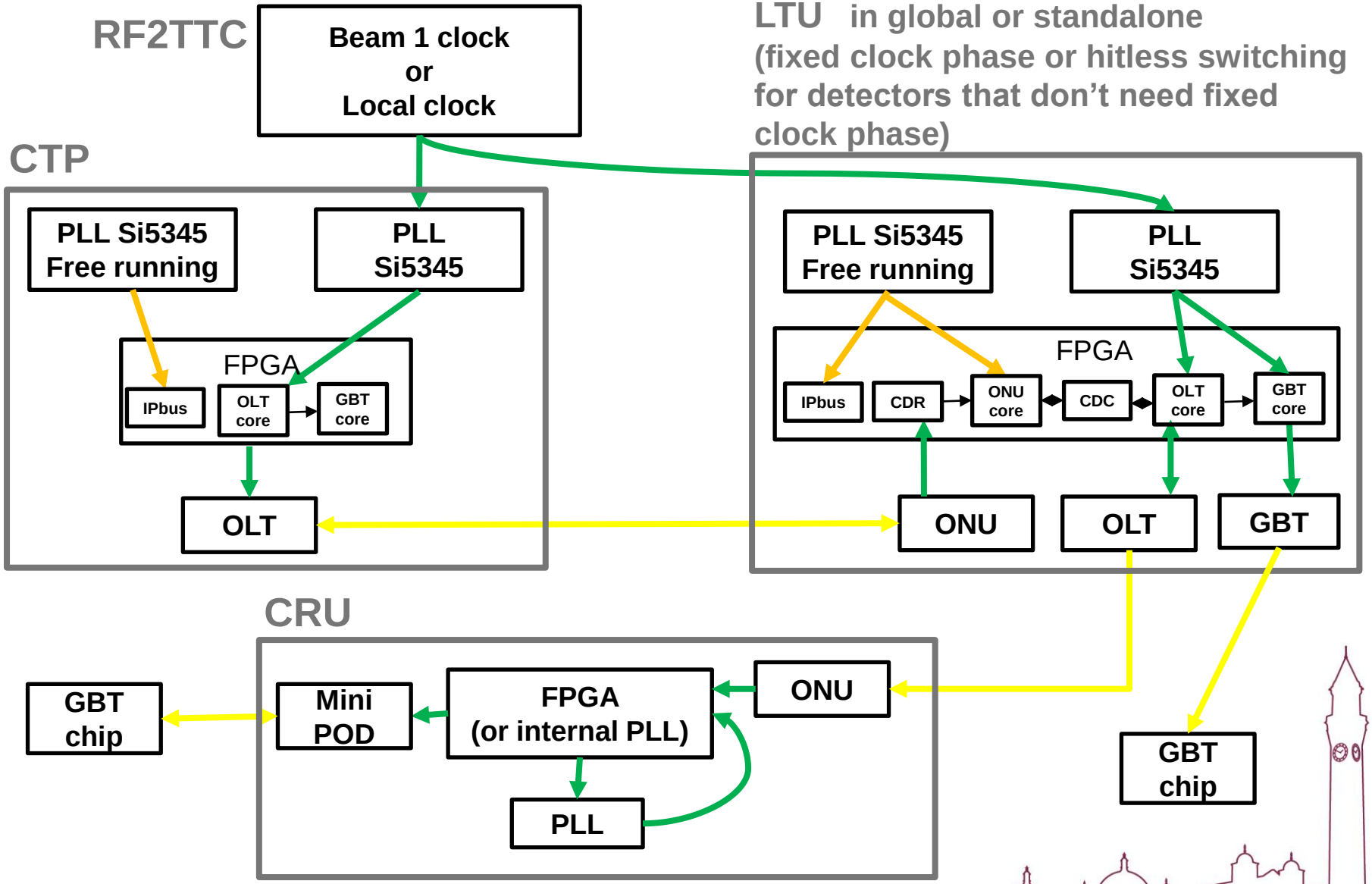
CTP core logic

Trigger class = Trigger Condition + Trigger Cluster + Trigger Vetoes

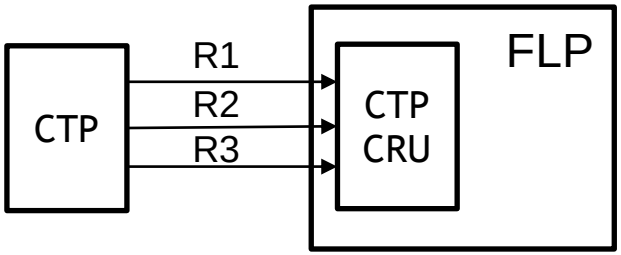
- **Trigger class** combines physics interest and readout
  - 64 classes
  - Important only to triggered detectors
- **Trigger Condition** defines a region of physics interest
  - Any logical function of the trigger inputs
  - Bunch crossing (BC) mask
    - usually corresponds to LHC filling scheme
      which BC are considered for interaction
- **Trigger Cluster**
  - Group of detectors to be read out - up to 18 different clusters
  - Defined by the HB function
- **Trigger Vetoes**
  - Cluster busy = detectors in cluster either in HBr state or in Busy state for triggered detectors
    - Allows a correlation between triggered and continuous detectors
      - » i.e no triggers in HBr frames



Overall clocking scheme



CTP readout



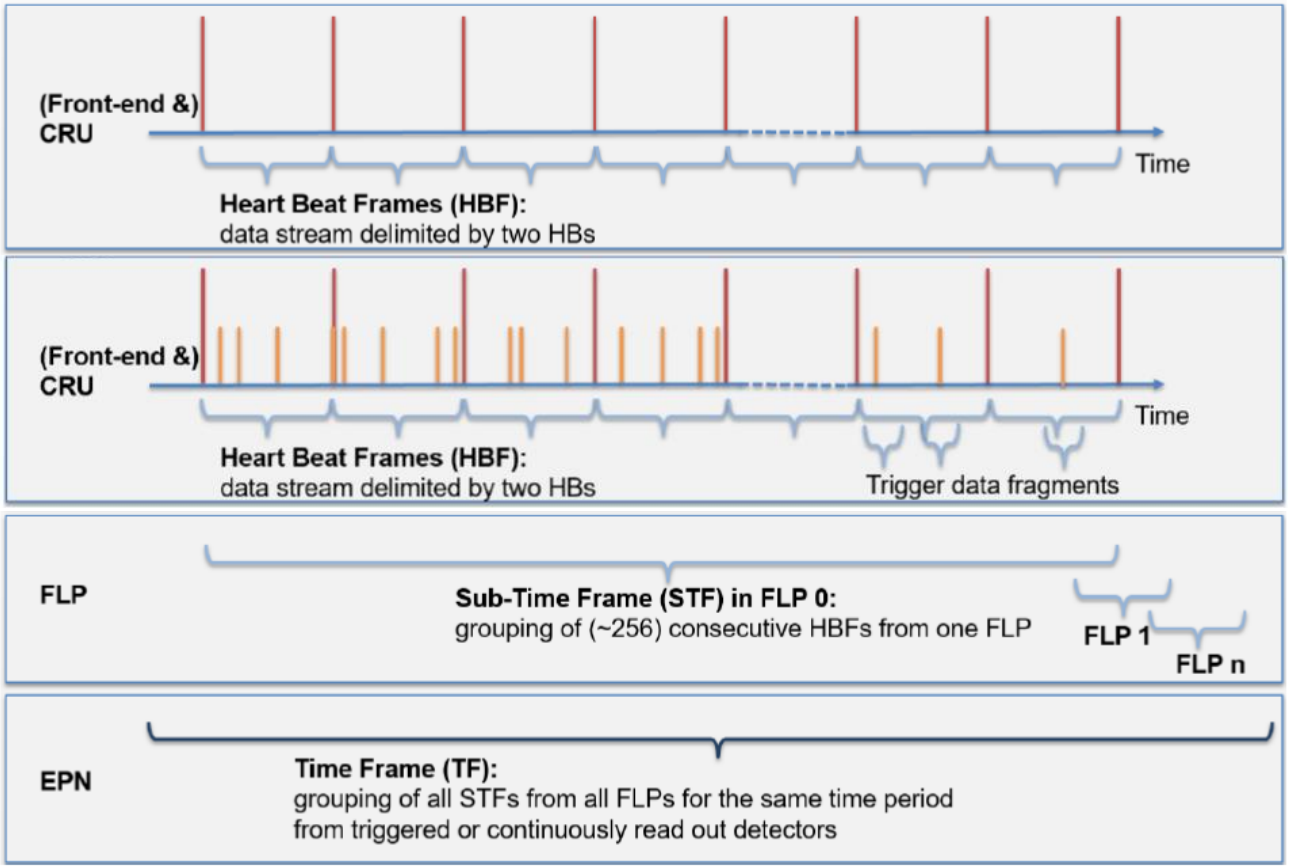
3 GBT links:
- R1 – Interaction record
- R2 – Trigger Class Mask
- R3 – Counters,
  HB Decision Record,
  HBs/BS maps

- Trigger Class Record (TCR)
  - Trigger Class Mask every BC – 64 bits
    - records what type of event was readout
    - with CTP Configuration => relation between Trigger Inputs, Trigger Class and Cluster => which triggered detectors are readout in given BC
- Interaction record (IR)
  - Trigger Input Mask every BC – 48 bits
    - Luminosity global and bunch by bunch
    - With CTP Configuration => cross check of consistency between TCR and IR

Continuous vs. triggered readout

- **Continuous readout is the main mode of operation**
- Detectors push **continuous stream of data** which are delimited by CTP HeartBeat (HB) triggers
- They must be capable of running in **triggered mode** as well

- ALICE data is divided into **HeartBeat frames** (HBf)
  - Each HBf is 88.92 μs - Time to fully read out TPC (length of LHC orbit)
  - 128 (programable) HBf compose a **Time-Frame** (TF)



Continuous readout

| HeartBeat Triggers
| Physics Triggers

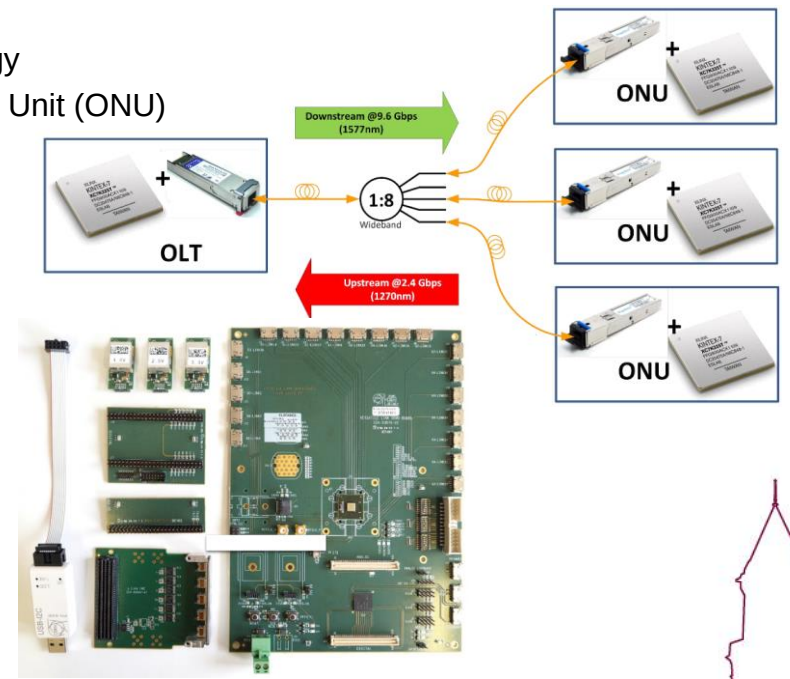
Triggered readout

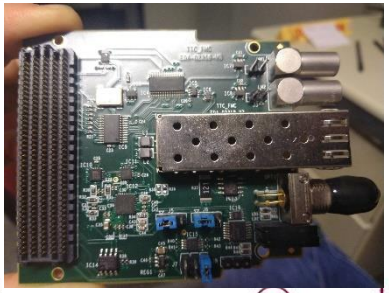
First Level Processor (FLP)
FLPs see only 3 CRU

Event Processing Node (EPN)
EPNs see all CRU

Transceivers

- **TTC-PON**
  - Off-the-shelf Passive Optical Network (PON) technology
    - Optical Line Terminal (OLT) and Optical Network Unit (ONU)
  - Bidirectional, up to 9.6 Gbps downstream
    - 200 user bits per bunch crossing
  - Communication between **CTP-LTU** and **LTU-CRU**
- **GBT**
  - Gigabit Transceiver
  - Radiation harnessed links
  - Bidirectional, up to 4.8 Gbps
    - 80 user bits per bunch crossing
  - Communication between **LTU-FEE** and **FEE-CRU**
- **TTC**
  - Trigger-Timing-Control developed by RD12 collaboration used till end of Run 2
  - Kept for backward compatibility for non-CRU detectors
  - 80 Mbps total downstream split in 2 channels (A and B)
    - synchronous trigger bit (in A) and asynchronous payload (in B)
  - Communication between **LTU-FEE** (legacy)





Trigger protocol

- **Trigger message** contains a time identification and a control/state (trigger type)
  - **Event Identification** – 44 bits
    - 32 bits LHC Orbit
    - 12 bits Bunch Crossing in a given Orbit
  - **Trigger Type** – 32 bits
    - Specify what happened in a given ID
    - Physics Trigger, Calibration, LHC Orbit, HeartBeat, HeartBeat reject, Start of Run, End of Run etc.
- **TTC-PON + GBT**
  - These 76 bits are sent each BC over PON and GBT
  - In addition PON also contains HB decision record
    - List of HB decisions in a given Time Frame
- **RD12 TTC**
  - 76 bits are asynchronously send over B channel by chopping into 7 TTC words (full transmission takes 308 BC)
    - Due to limited bandwidth only relevant control/states for particular detector are transmitted
      - Physics Trigger, Calibration, Start of Run, End of Run
      - Orbit and Calibration request require channel B resynchronisation with LHC and are broadcasted as short message of 16 bits

Trigger Types

Bit	Name	Comment
0	ORBIT	ORBIT
1	HB	Heart Beat flag
2	HBr	Heart Beat reject flag
3	HC	Health Check
4	PhT	Physics Trigger
5	PP	Pre Pulse for calibration
6	Cal	Calibration trigger
7	SOT	Start of Triggered Data
8	EOT	End of Triggered Data
9	SOC	Start of Continuous Data
10	EOC	End of Continuous Data
11	TF	Time Frame delimiter
12	FErst	Front End reset
13	RT	Run Type; 1=Cont, 0=Trig
14	RS	Running State; 1=Running
...	...	Spare
27	LHCgap1	LHC abort gap 1
28	LHCgap2	LHC abort gap 2
29	TPCsync	TPC synchronisation/ITSrst
30	TPCrst	On request reset
31	TOF	TOF special trigger

PON data format Trigger Message

PON bit	PON byte	Payload	Content
<31:0>	0-3	<31:0>	Trigger Type
<43:32>	4-5	<11:0>	BCID
<47:44>	5	<3:0>	Trigger Level/Spare
<79:48>	6-9	<31:0>	ORBIT
<118:80>	10-14	<38:0>	spare
<119:119>	14	<0:0>	TTValid
<120:120>	15	<0:0>	Header Flag
<127:121>	15	<6:0>	Word Count
<143:128>	16-17	<15:0>	HBDR payload
<144:144>	18	<0:0>	HBDRValid
<198:145>	18-24	<54:0>	Spare