RD53 Status and the Role of Verification in Digital Design

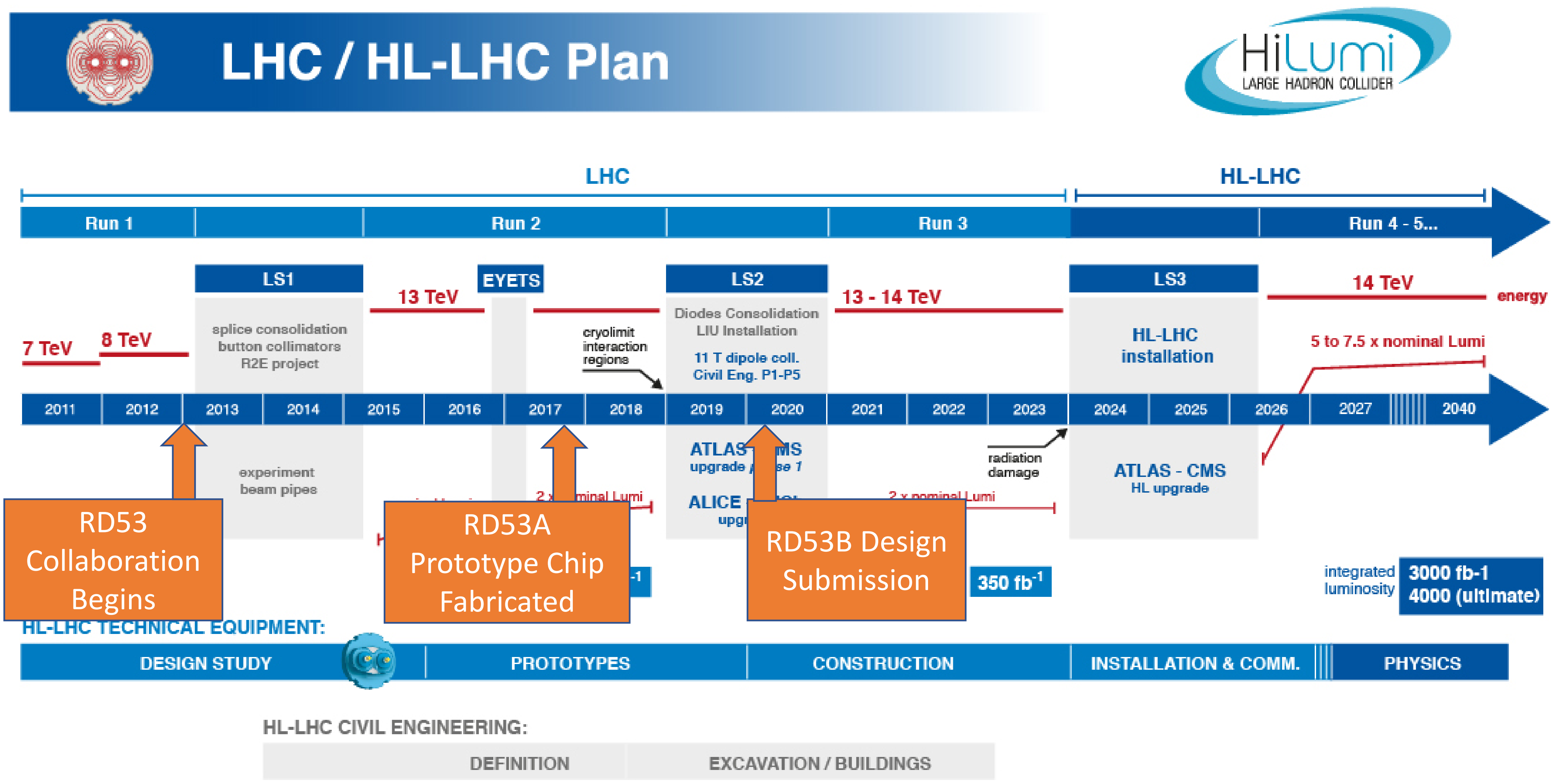
- CPAD 2019 Dec. 8-10, 2019

- Cesar Gonzalez Renteria (LBNL) on behalf of the RD53 Collaboration

Outline

- Introduction
- RD53 Collaboration
  - What is the mission of the collaboration and what is the final result
- Verification
  - What is verification and how is it used
- Verification of the RD53B Chip
  - Structure of the verification workbench in the case of RD53B

LHC / HL-LHC Plan

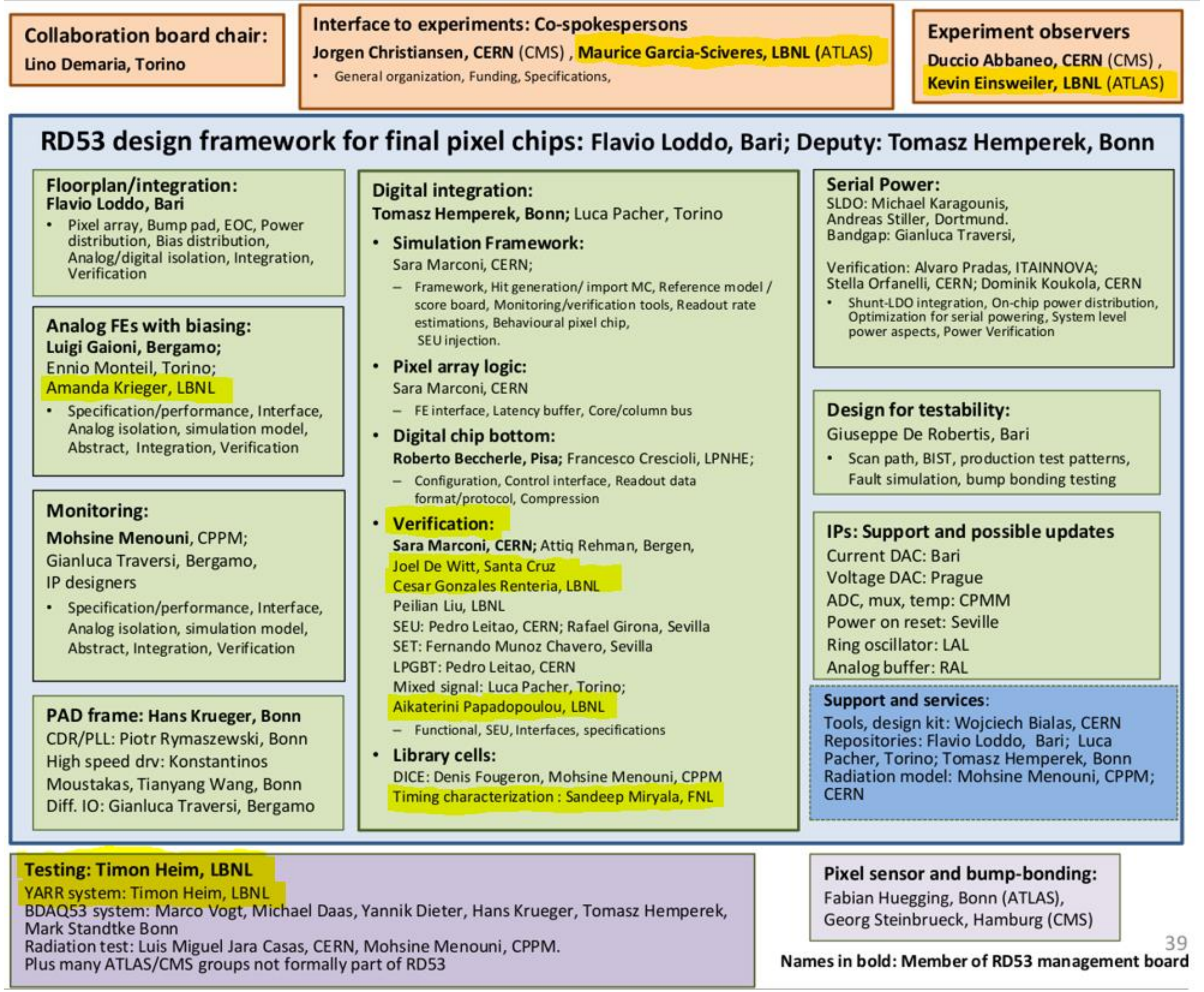


RD53 Collaboration

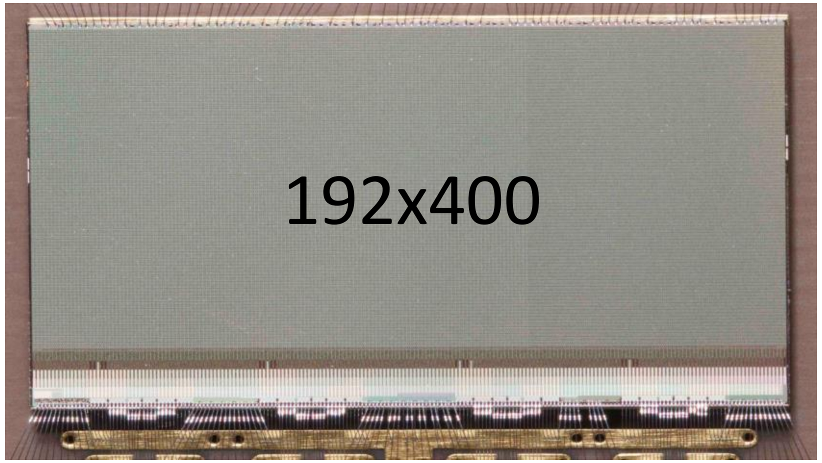
- Focused R&D to develop pixel chips for both ATLAS and CMS upgrades
- Established recognizing that HL-LHC pixel requirements were extremely challenging, yet very similar for both experiment, and a joint effort was the best way to meet them
- At the request of the experiments, last year the mandate of RD53 was extended to design the final production chips for ATLAS and CMS
  - Keep the design team together.
  - Pursue as much as possible a common design to serve the needs of both experiments.
- RD53 has 22 collaborating institutes and many Guests
  - Roughly 20 designers on RD53B Chip (next slide)
  - ~100 conference talks/proceedings/papers to date

RD53 Collaboration

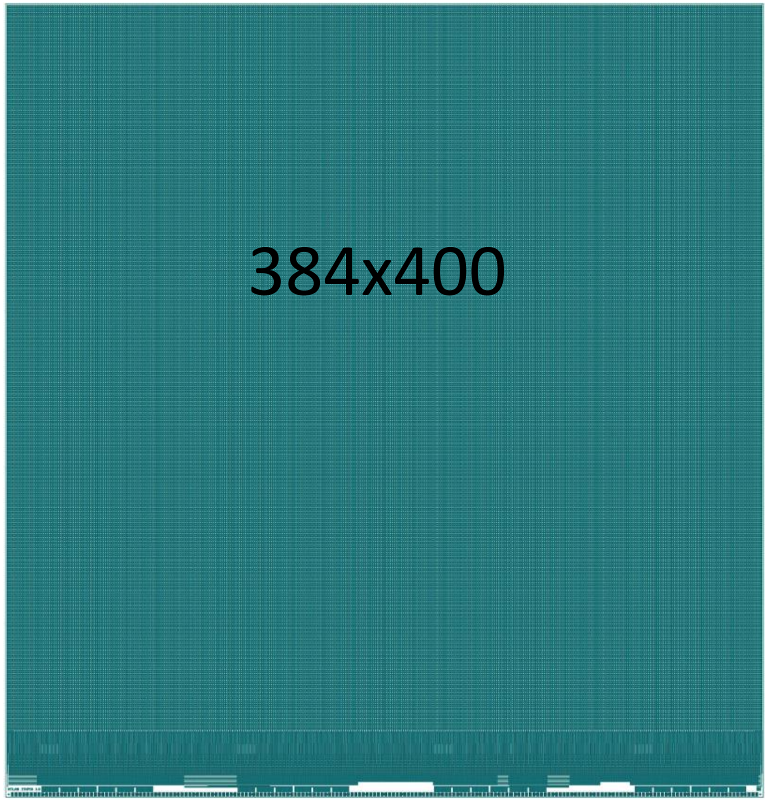
- US Institutes:
  - Lawrence Berkeley National Laboratory
  - University of California, Santa Cruz
  - Fermi National Laboratory
  - University of New Mexico



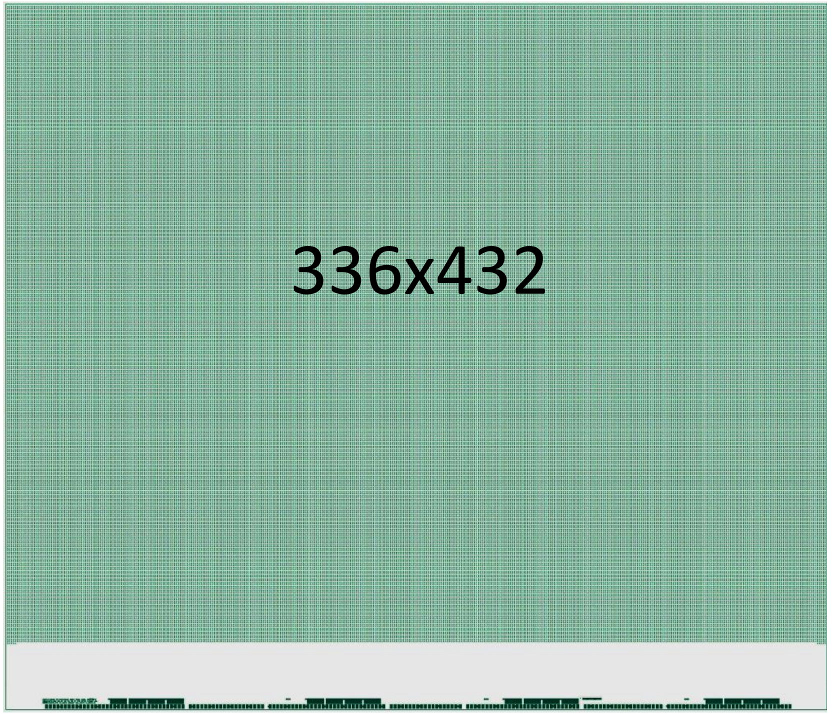
Picture of RD53A, B-ATLAS, B-CMS



RD53A Chip

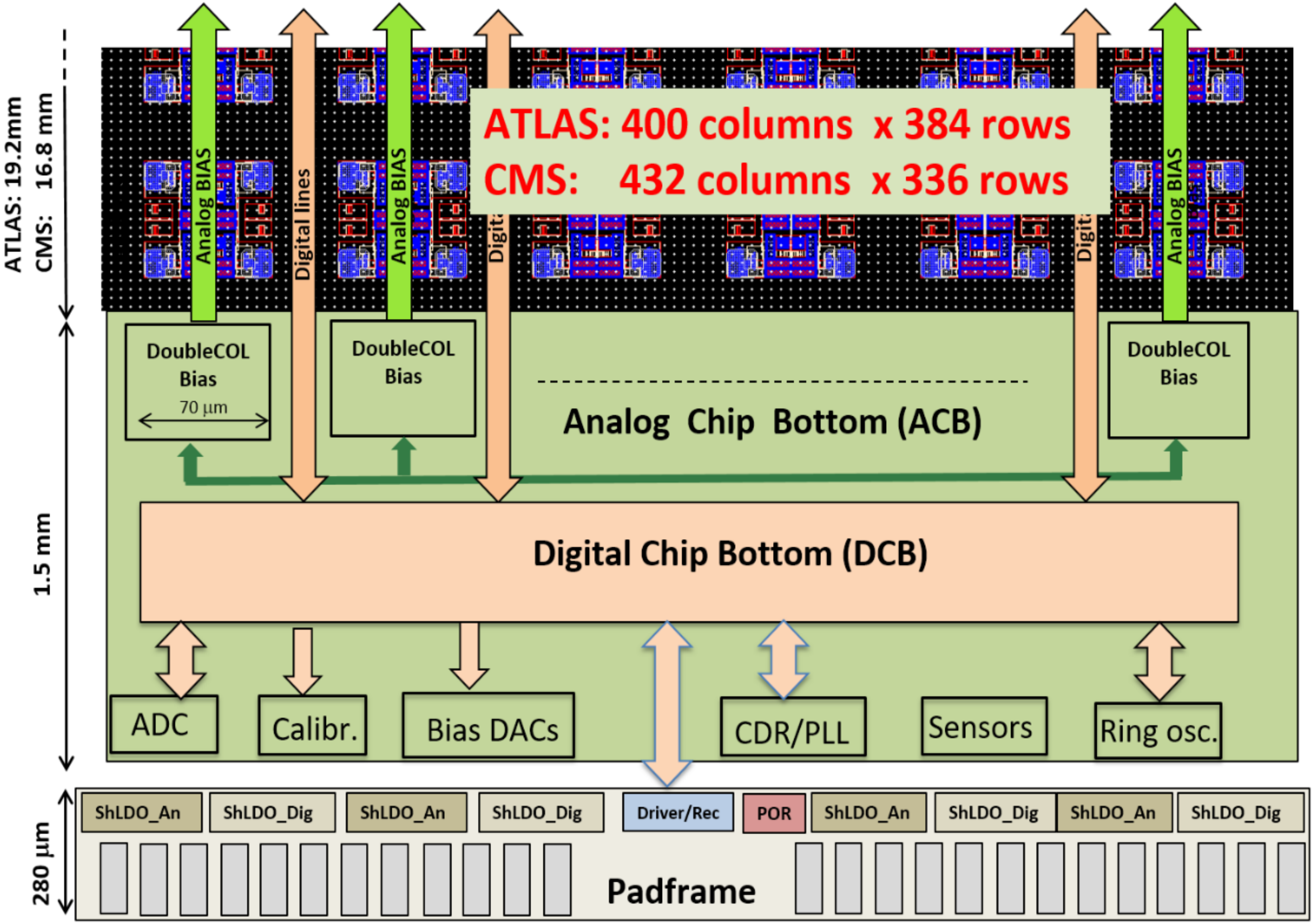


RD53B-ATLAS Chip



RD53B-CMS Chip

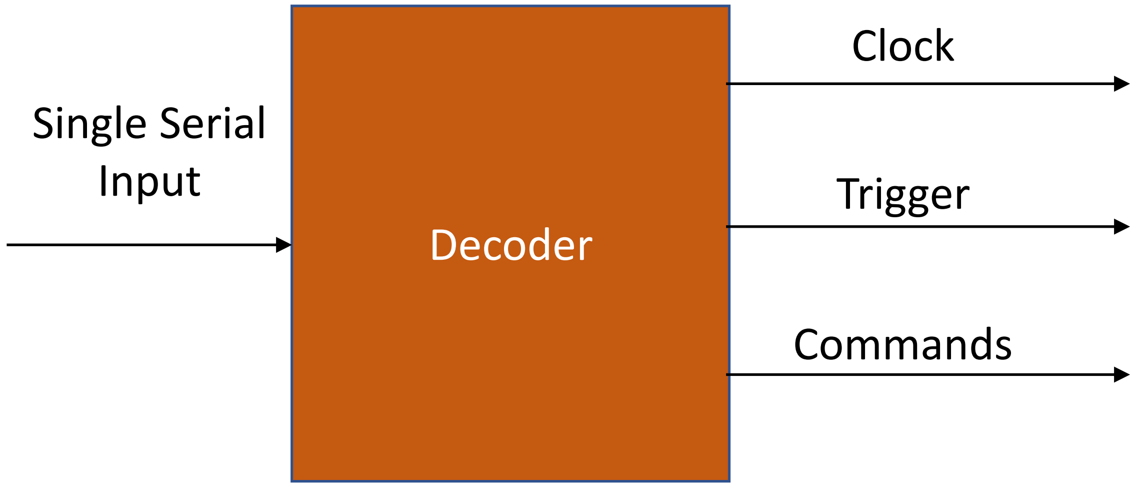
RD53B Chip Design



- Read-Out ASIC for both experiments which communicates with Pixel Matrix and DAQ system
- Analog portion of chip done the standard way.
- Digital part is extremely complicated and makes up most of the design.

Example of Digital Complexity: I/O

- Three main signals necessary for chip operation:
  - Bunch Crossing **Clock**
  - **Trigger** (Synchronous)
  - **Commands** (Configuration)
- In order to eliminate massive amounts of service lines, designed a scheme that sends all three in one serial stream.



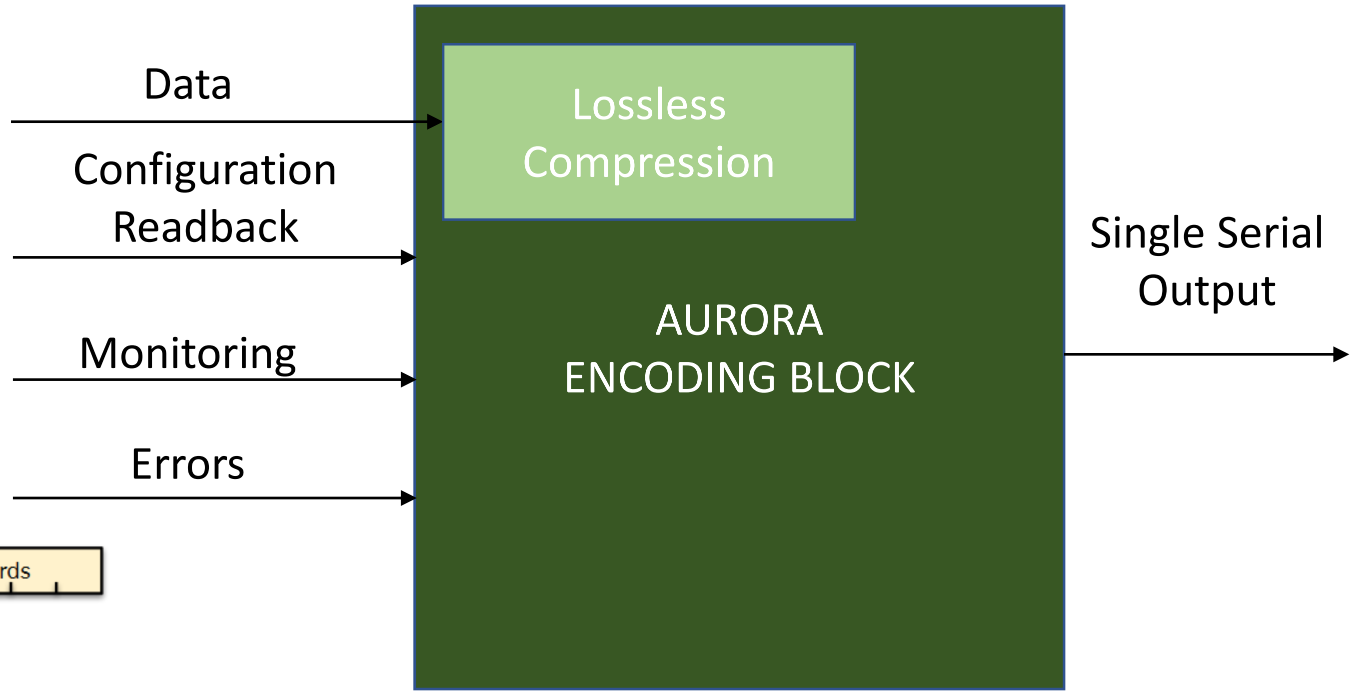
Structure of the Serial Stream

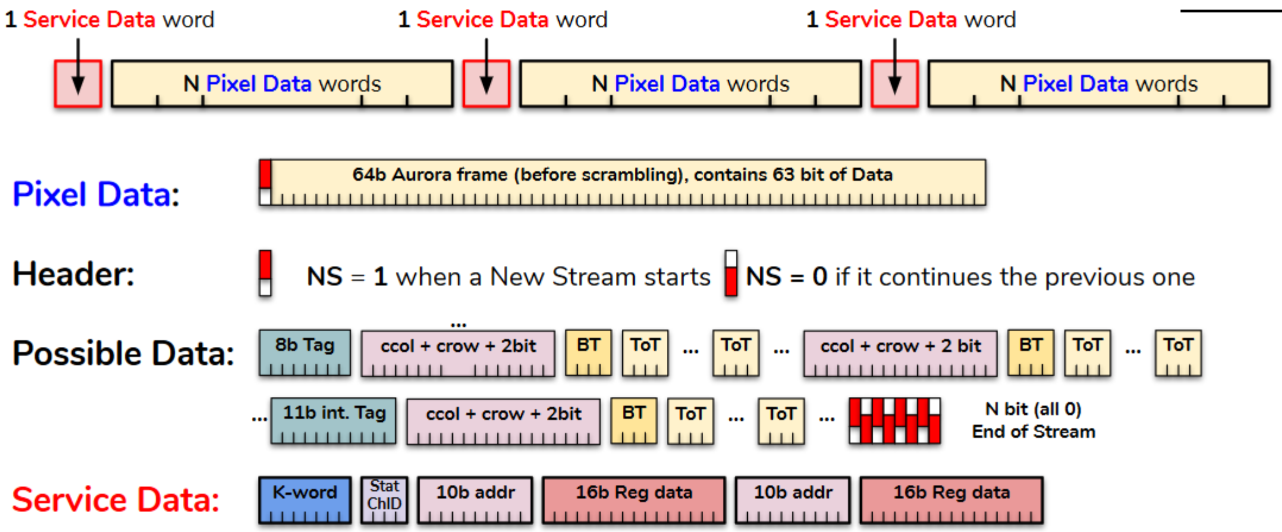
- Designed encoding scheme to make sure signals are synchronous to BC Clock, DC Balanced (AC coupling compatible due to serial power)
- Trigger/Commands sent in 16 bit frames made of 2 8-bit symbols
- Each Trigger/Command has a encoded symbol which defines it (52 total encoded symbols)
- Will comment later on how chip decodes this bit stream

Command	Encoding		(T)ag, (A)ddress or (D)ata 5-bit content					
Sync	1000_0001	0111_1110						
PLLlock (noop)	1010_1010	1010_1010						
Trigger	tttt_tttt	Tag[0..53]						
Read_trigger	0110_1001	ID<4:0>	00,T<7:5>	T<4:0>				
Clear	0101_1010	ID<4:0>						
Global Pulse	0101_1100	ID<4:0>						
Cal	0110_0011	ID<4:0>	D<19:15>	D<14:10>	D<9:5>	D<4:0>		
WrReg(0)	0110_0110	ID<4:0>	0,A<8:5>	A<4:0>	D<15:11>	D<10:6>	D<5:1>	D<0>,0000
WrReg(1)	0110_0110	ID<4:0>	1,A<8:5>	A<4:0>	N×(D<9:5>	D<4:0>)		
RdReg	0110_0101	ID<4:0>	0,A<8:5>	A<4:0>				

Output: Aurora Encoding

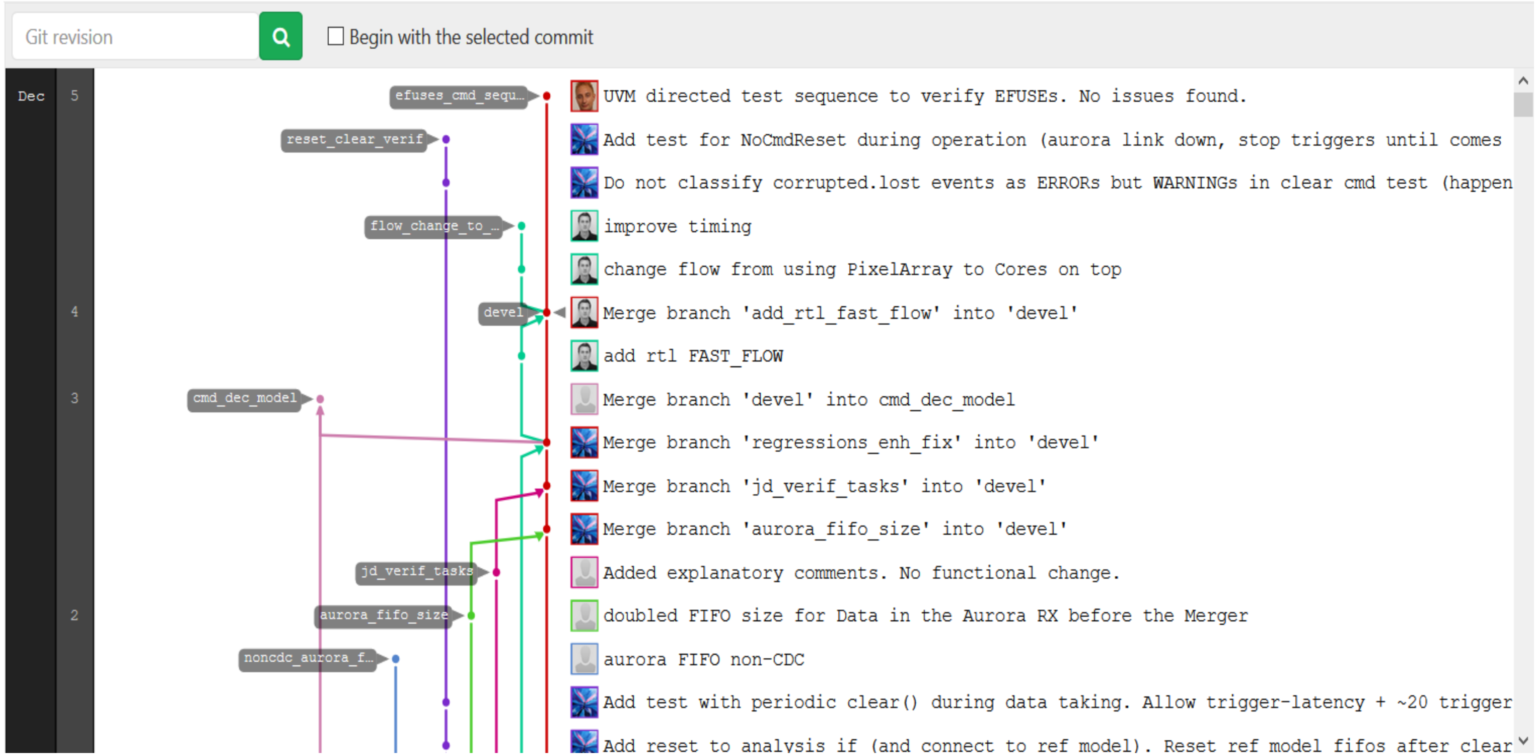
- Output to DAQ is asynchronous
- Encode multiple signals using Aurora 64b/66b encoding.
- Industry standard encoding protocol similar to ethernet protocol
- Makeup of Aurora Frame designed to be efficient





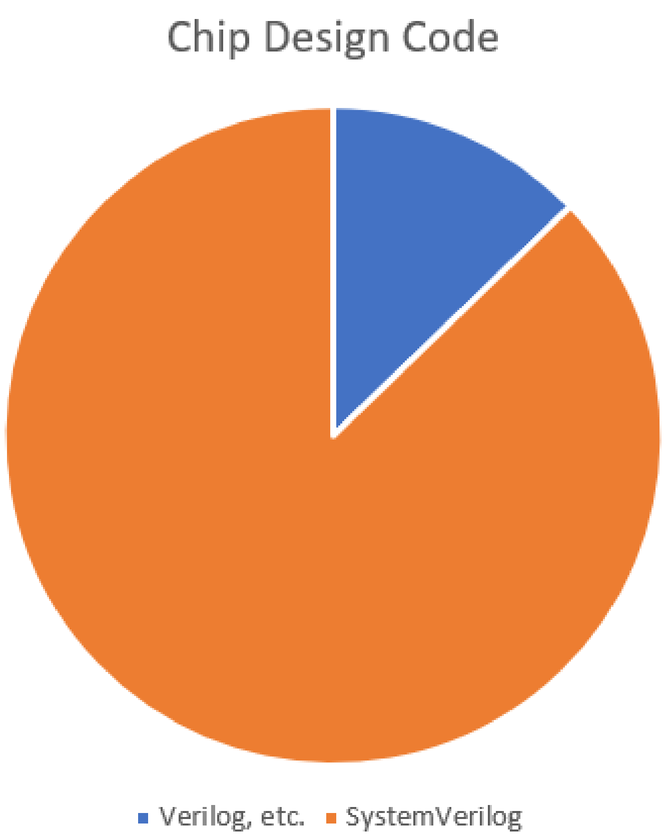
Digital Chip Design is a Software Project

- **System Verilog** is a hardware verification and description language similar to C++ but with built in abstractions for hardware objects, ability to **manipulate time**, and dynamically allocate memory.
- Built from **Verilog** which was similar to C. Verilog is missing C++-style objects, assertions and coverage groups.
- **UVM (Universal Verification Methodology)** is a pre-built library written in SV with a full verification testbench structure in place inherited by UVM classes.
- Through Verilog, SV & UVM you can:
  - Write up the logic and parts that make up the chip (DUT).
  - Write up the testbench to stimulate the DUT and analyze the results.



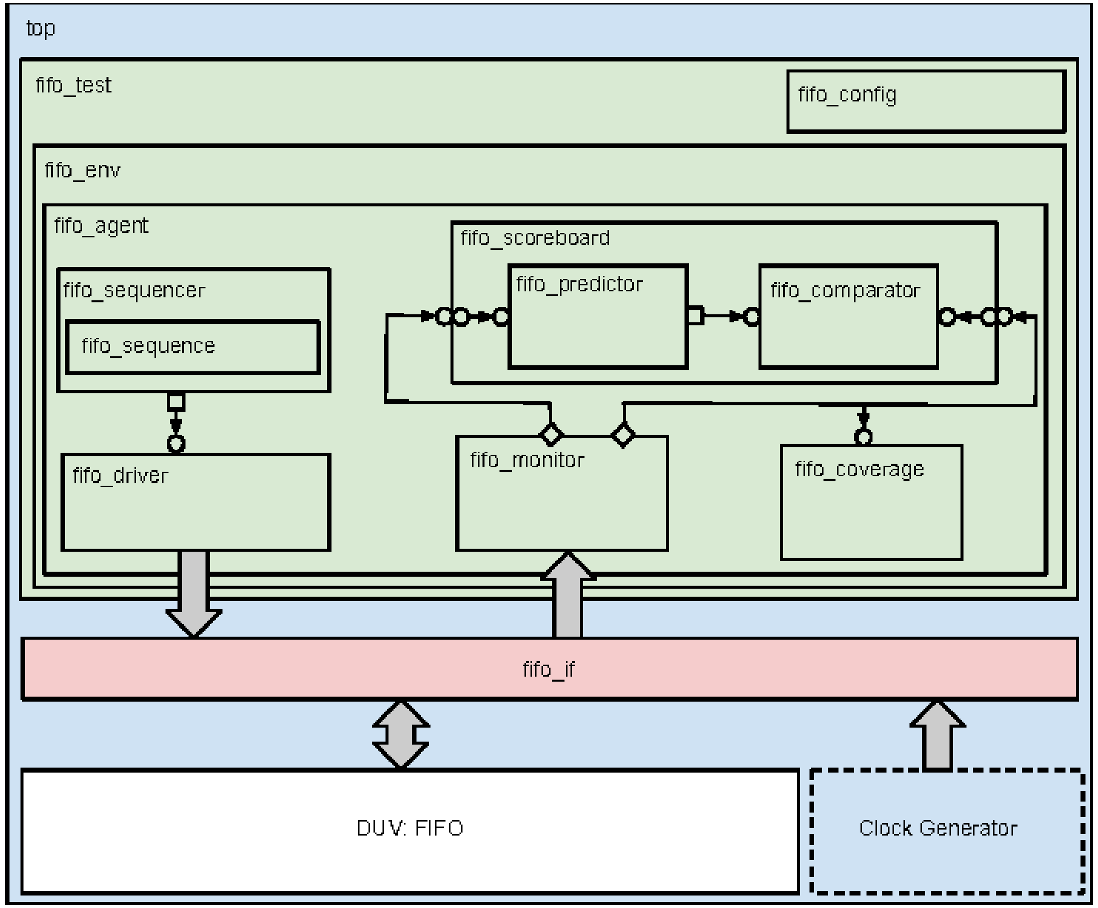
** The Chip Design for RD53B is all software!!**

- Digital Code in the Chip:
- There are 123 different SystemVerilog files and each file corresponds (mostly) to one module.
- 87% of files written in SystemVerilog
- There are ~20k lines of code.
- Verification Code:
- There are 392 different SystemVerilog files
- 90% of files written in SystemVerilog
- There are ~120k lines of code.



Example: Verifying a FIFO Design

- The top of the workbench is the **test**.
- Sample test can be as follows:
  - Write to FIFO (**Driver**)
  - Check state and size of FIFO before and after write (**Monitor**)
  - Read from FIFO (**Driver**)
  - Check state and size of FIFO before and after read (**Monitor**)
  - Verify it matches expectation (**Scoreboard**)
    - I.E. First In – First Out

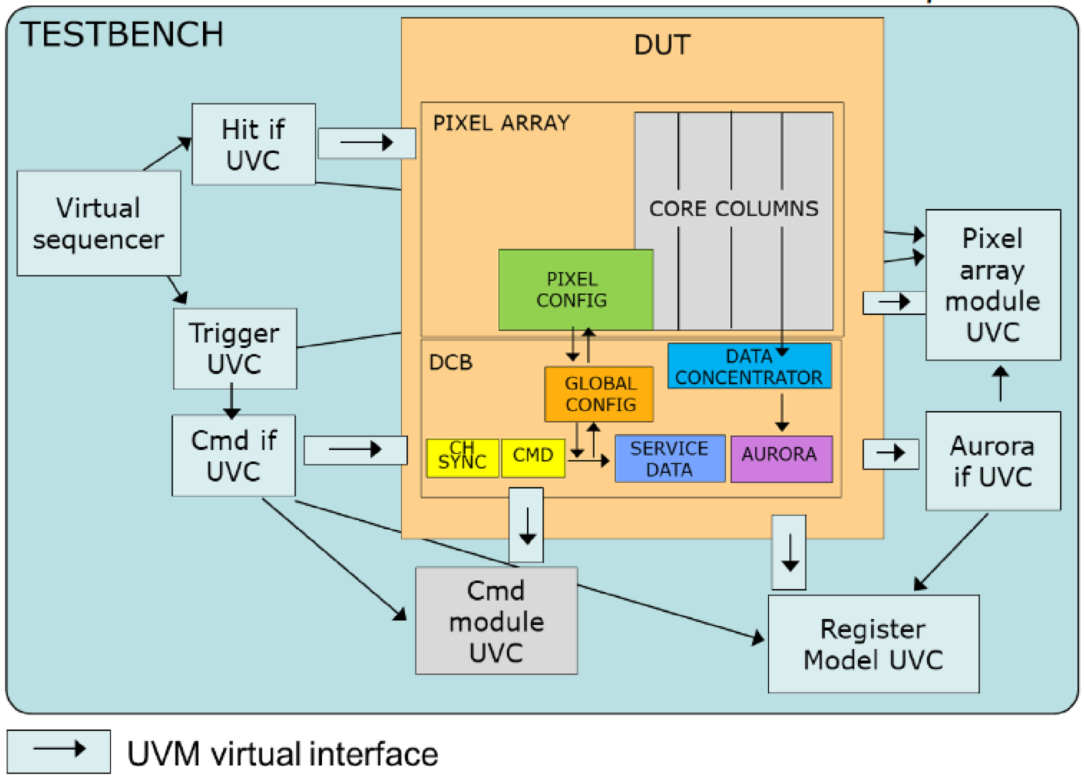


[3]

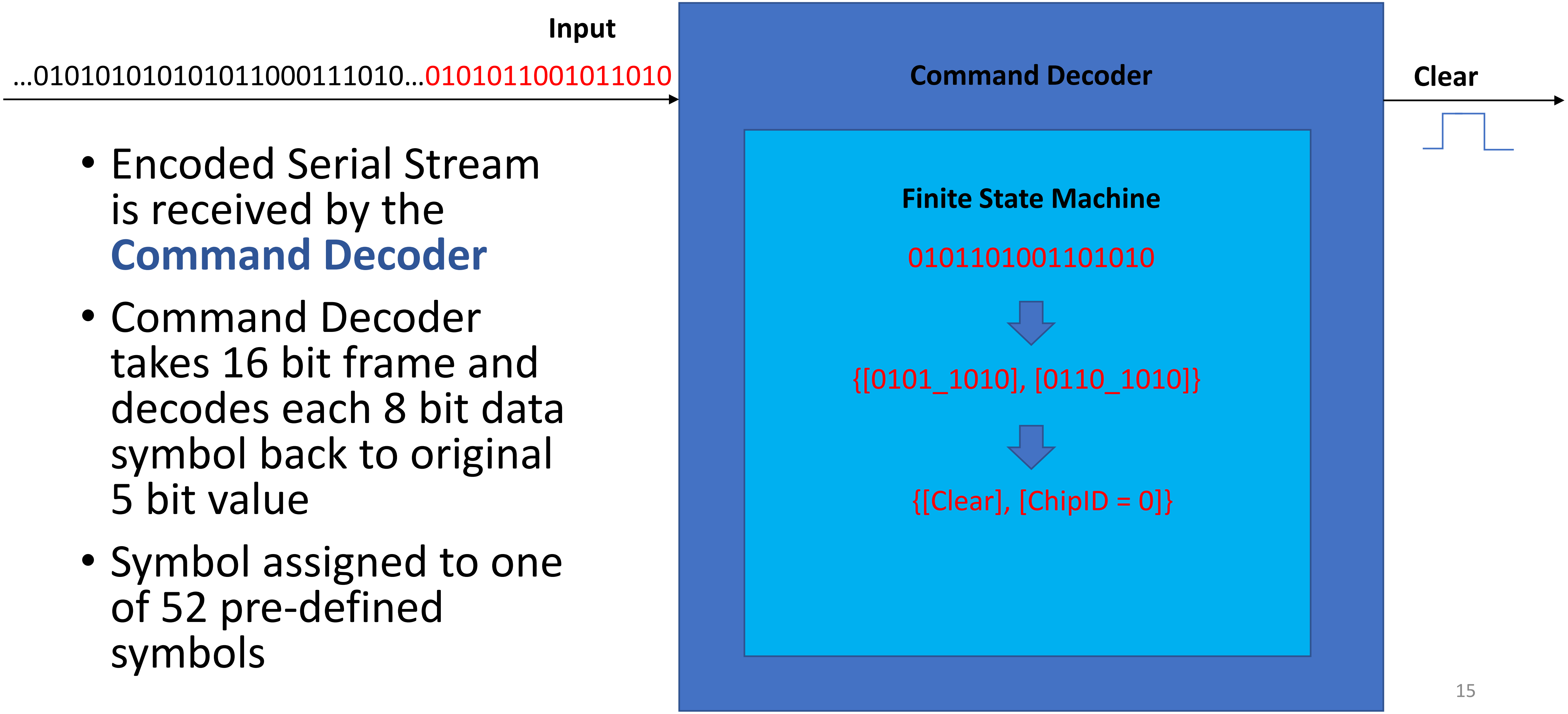
RD53B Chip UVM Testbench

- **Top Test** instantiates multiple Universal Verification Components (UVCs).
- Each UVC is created to test certain blocks of the chip design.
- Each UVC contains its own environment, agent, scoreboard, etc.
- Will show an example of verifying the “CMD” block of the chip

UVC = Universal Verification Component

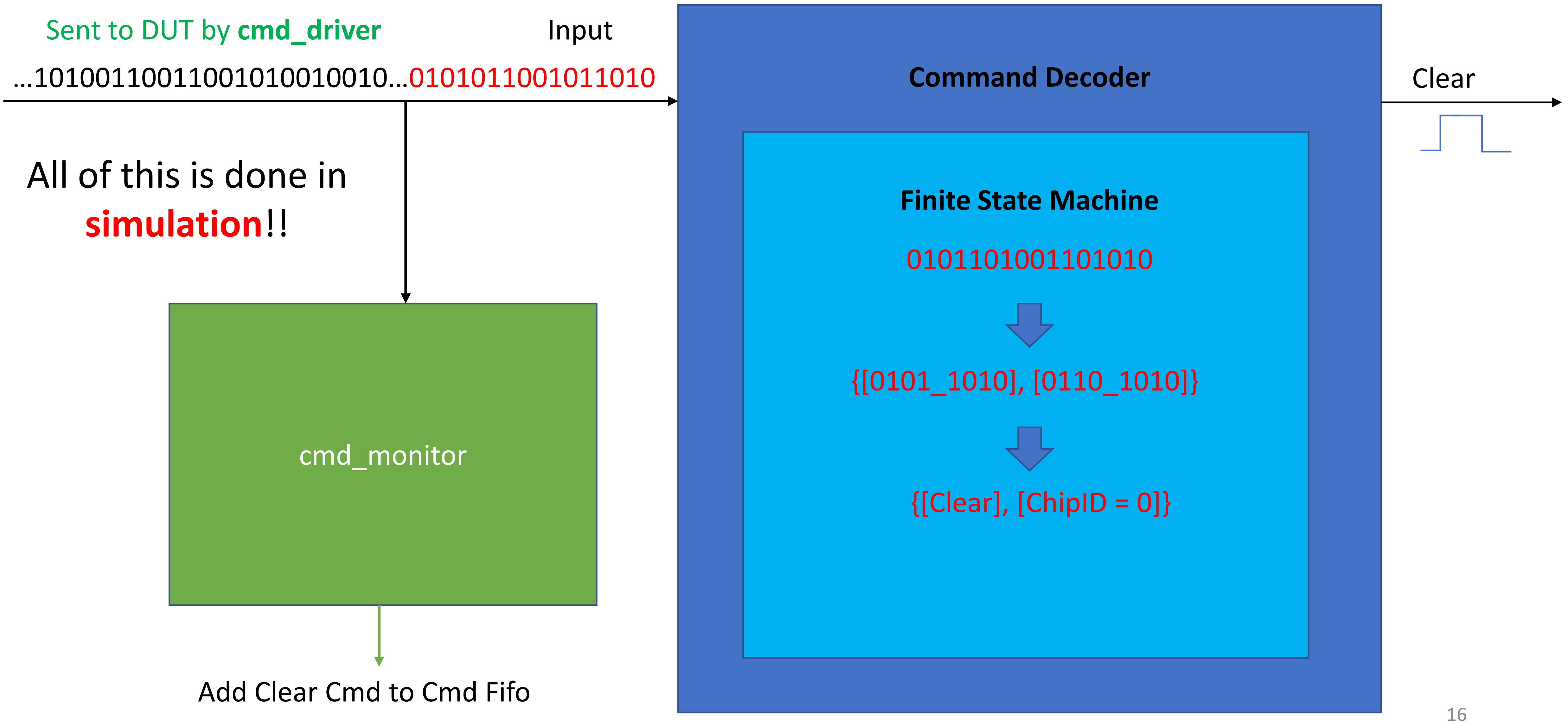


An Example: Verifying the Command Decoder

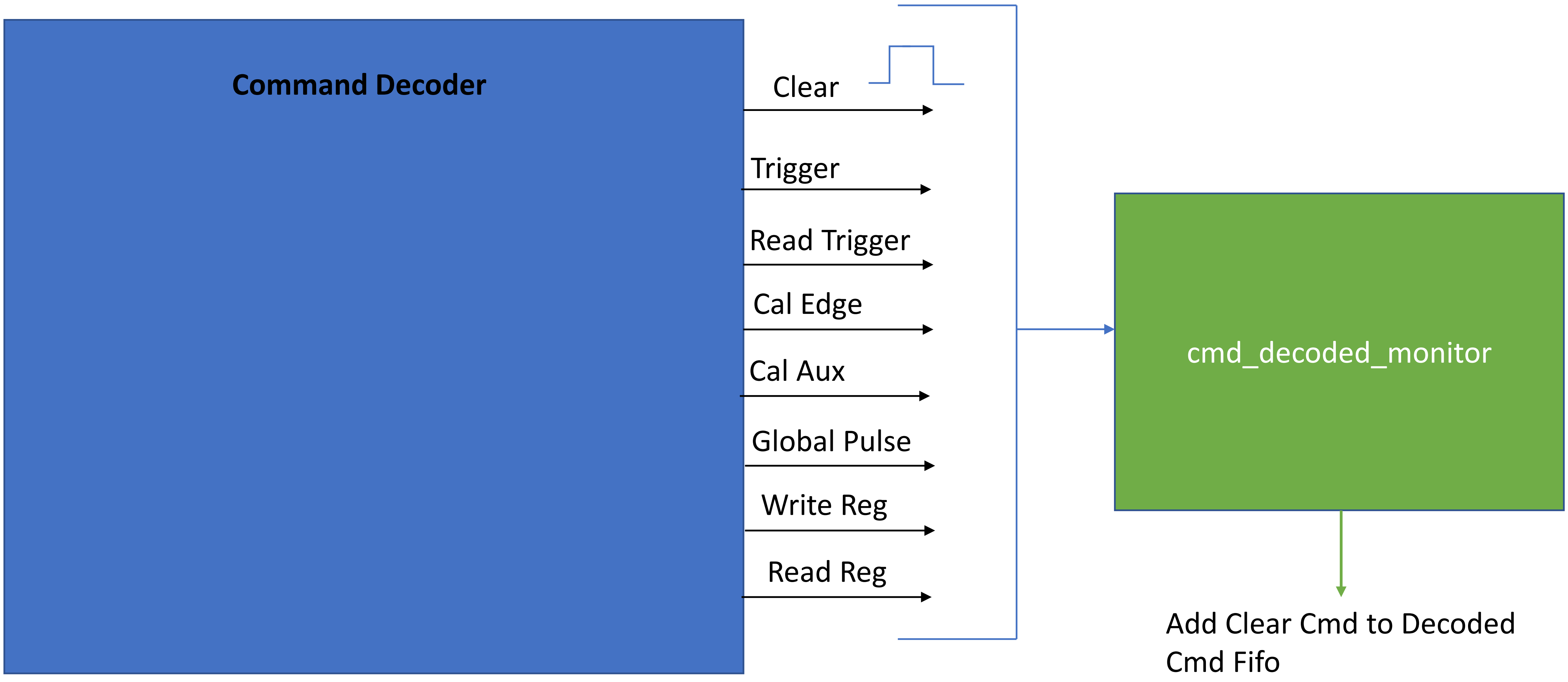


- Encoded Serial Stream is received by the **Command Decoder**
- Command Decoder takes 16 bit frame and decodes each 8 bit data symbol back to original 5 bit value
- Symbol assigned to one of 52 pre-defined symbols

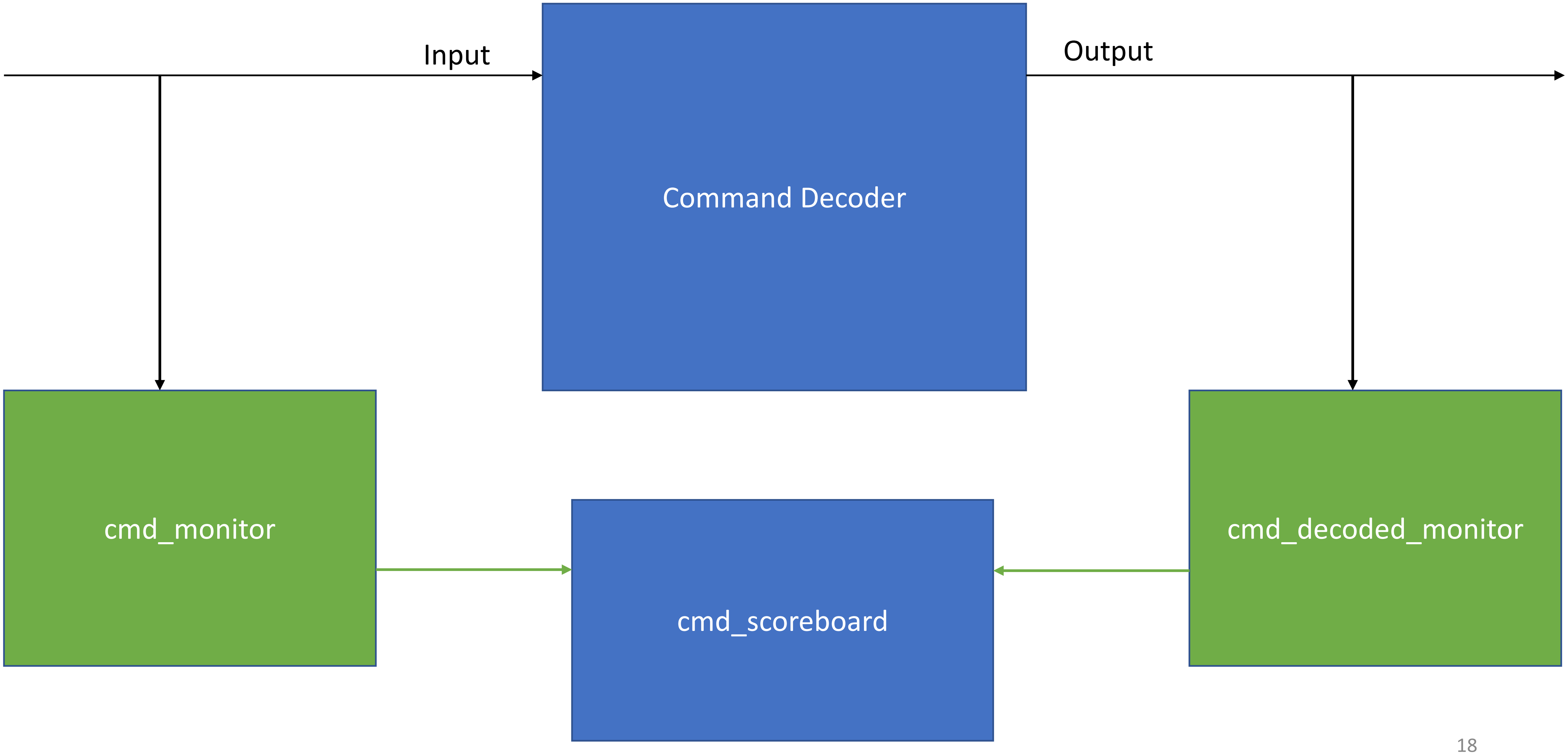
Introducing Verification Structure: UVM Monitor and Driver



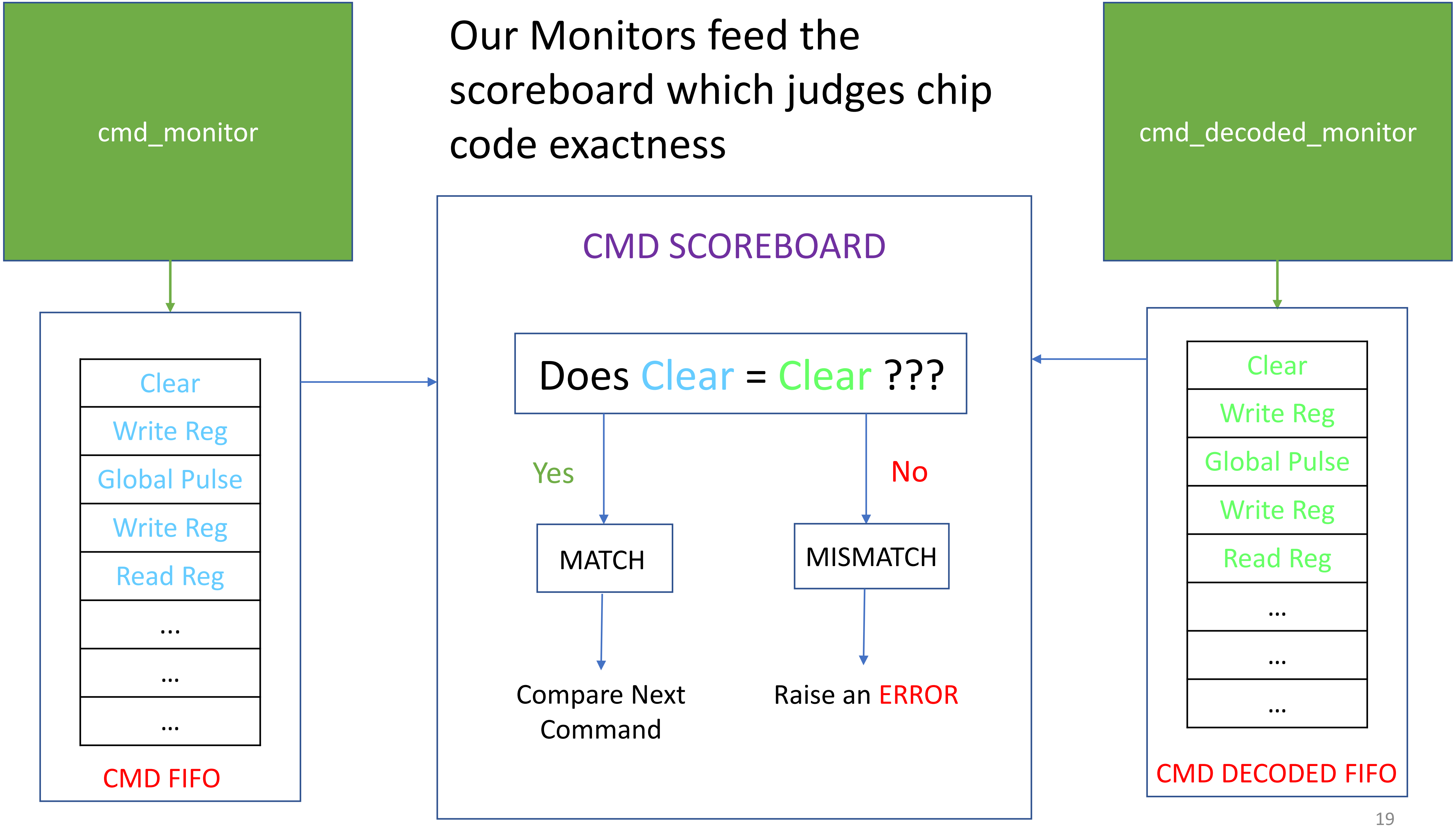
Introducing the Second UVM Monitor



Testing Correct Behavior: UVM Scoreboard



Our Monitors feed the scoreboard which judges chip code exactness

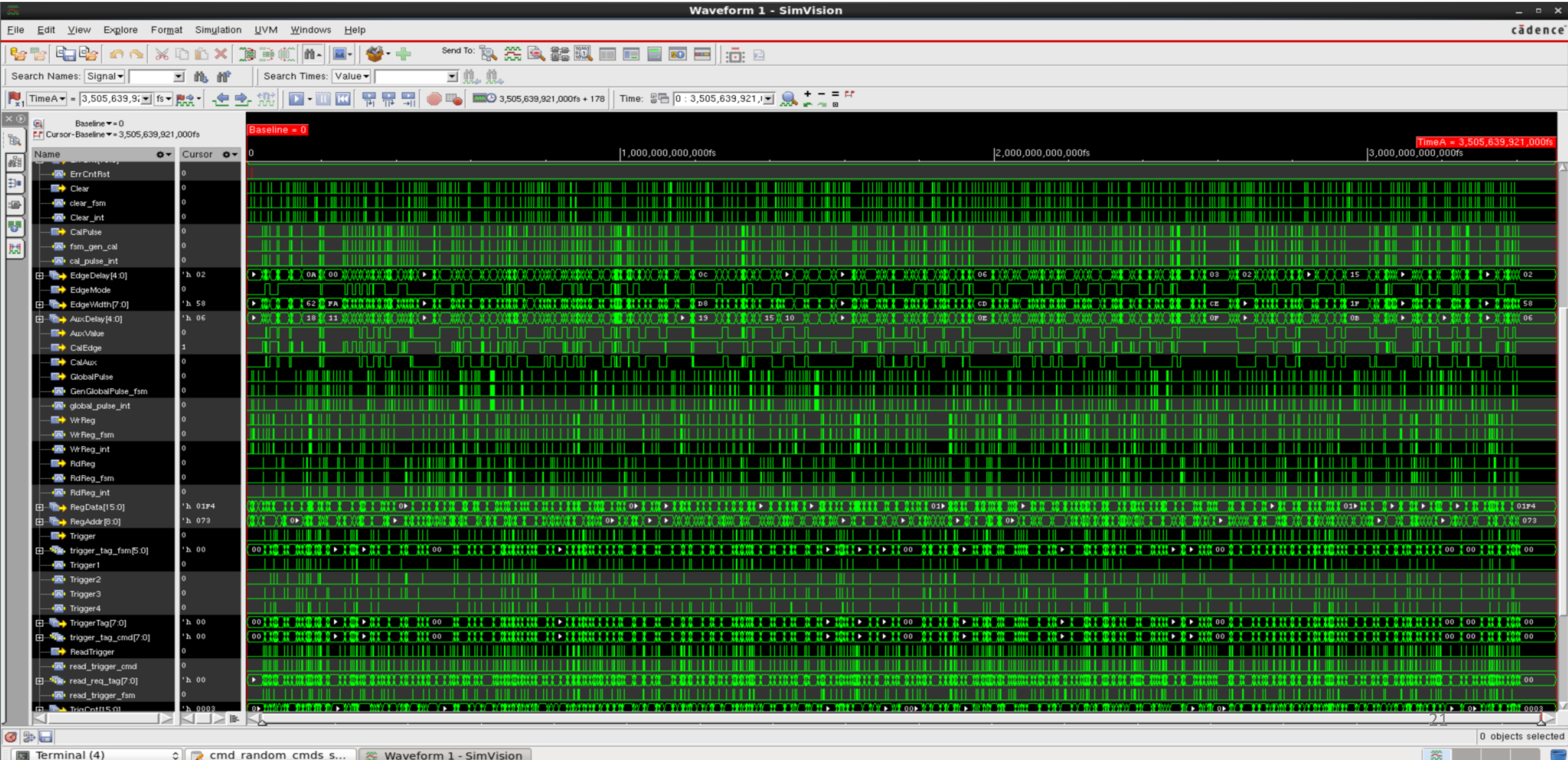


Create a Test: Do we see a match?

- 2000 Randomly generated commands sent + extra commands to set up chip
- The data sent with each command is also randomly generated (within allowable values)
- Command Checker finds a match between Decoded and Encoded monitors for every command sent.

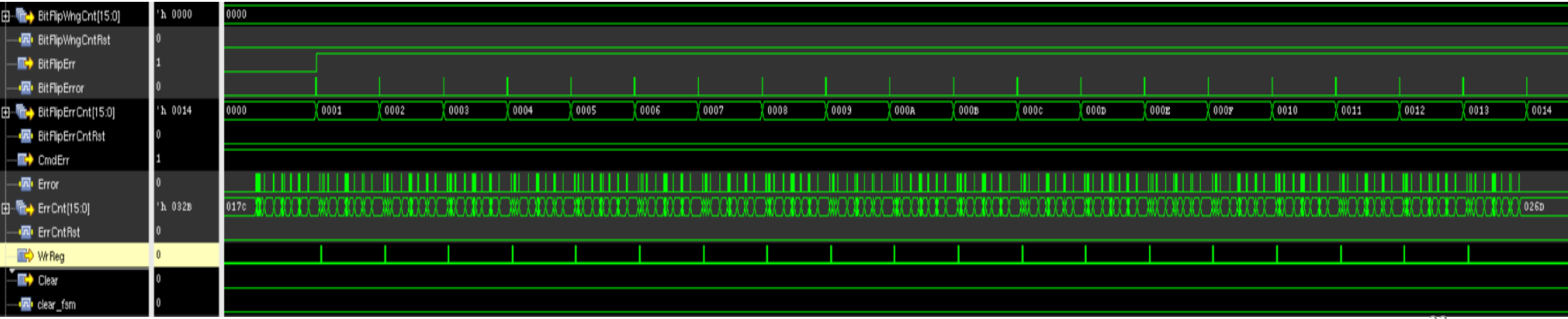
```
UVM_INFO
/home/cesar_gr/RD53B/work/sim/../../cds/../sim/verification_environment/cmd/sv/cmd_checker.sv(102)
@ 38746145000: uvm_test_top.m_env.m_cmd_analysis_env.m_cmd_checker [CMD_CHECKER_MATCH] WRITE
REGISTER        Chip #31,         no auto-increment, address EnCoreColumnCalibration_1,    data e1 encoded cmd
matches decoded cmd WRITE REGISTER        Chip #31,         no auto-increment, address
EnCoreColumnCalibration_1,        data e1

UVM_INFO
/home/cesar_gr/RD53B/work/sim/../../cds/../sim/verification_environment/cmd/sv/cmd_checker.sv(102)
@ 38896289000: uvm_test_top.m_env.m_cmd_analysis_env.m_cmd_checker [CMD_CHECKER_MATCH]
TRIGGER Positions of trigger during frame: 0001; trigger tag: 25 encoded cmd matches decoded cmd
TRIGGER Positions of trigger during frame: 0001; trigger tag: 25

UVM_INFO
/home/cesar_gr/RD53B/work/sim/../../cds/../sim/verification_environment/cmd/sv/cmd_checker.sv(102)
@ 39246625000: uvm_test_top.m_env.m_cmd_analysis_env.m_cmd_checker [CMD_CHECKER_MATCH] WRITE
REGISTER        Chip #31,         no auto-increment, address DAC_PREAMP_T_DIFF,    data 15 encoded cmd matches
decoded cmd WRITE REGISTER        Chip #31,         no auto-increment, address DAC_PREAMP_T_DIFF,    data 15

UVM_INFO
/home/cesar_gr/RD53B/work/sim/../../cds/../sim/verification_environment/cmd/sv/cmd_checker.sv(102)
@ 39446817000: uvm_test_top.m_env.m_cmd_analysis_env.m_cmd_checker [CMD_CHECKER_MATCH] READ
REGISTER        Chip #31,         address RingOscRoute encoded cmd matches decoded cmd READ REGISTER     Chip #31,
address RingOscRoute

UVM_INFO
/home/cesar_gr/RD53B/work/sim/../../cds/../sim/verification_environment/cmd/sv/cmd_monitor.sv(282)
@ 39596961000: uvm_test_top.m_env.m_cmd_env.m_cmd_master_agent.m_cmd_monitor [COMMAND_MONITORING]
TRIGGER Positions of trigger during frame: 0001; trigger tag: 49

UVM_INFO
/home/cesar_gr/RD53B/work/sim/../../cds/../sim/verification_environment/cmd/sv/cmd_checker.sv(102)
@ 39647009000: uvm_test_top.m_env.m_cmd_analysis_env.m_cmd_checker [CMD_CHECKER_MATCH] READ
REGISTER        Chip #31,         address DAC_PREAMP_TR_LIN encoded cmd matches decoded cmd READ REGISTER        Chip
#31,     address DAC_PREAMP_TR_LIN

UVM_INFO
/home/cesar_gr/RD53B/work/sim/../../cds/../sim/verification_environment/cmd/sv/cmd_checker.sv(102)
@ 39797153000: uvm_test_top.m_env.m_cmd_analysis_env.m_cmd_checker [CMD_CHECKER_MATCH]
TRIGGER Positions of trigger during frame: 0001; trigger tag: 49 encoded cmd matches decoded cmd
TRIGGER Positions of trigger during frame: 0001; trigger tag: 49

UVM_INFO
/home/cesar_gr/RD53B/work/sim/../../cds/../sim/verification_environment/cmd/sv/cmd_checker.sv(102)
@ 40047393000: uvm_test_top.m_env.m_cmd_analysis_env.m_cmd_checker [CMD_CHECKER_MATCH] CAL        Chip
#31,     calEdge mode: 1, calEdge delay: 3, calEdge width: 85, calAux mode: 1, calAux delay: 23
encoded cmd matches decoded cmd CAL        Chip #31,         calEdge mode: 1, calEdge delay: 3, calEdge width:
85, calAux mode: 1, calAux delay: 23

UVM_INFO
/home/cesar_gr/RD53B/work/sim/../../cds/../sim/verification_environment/cmd/sv/cmd_monitor.sv(282)
@ 42399649000: uvm_test_top.m_env.m_cmd_env.m_cmd_master_agent.m_cmd_monitor [COMMAND_MONITORING]
TRIGGER Positions of trigger during frame: 1011; trigger tag: 23

UVM_INFO
/home/cesar_gr/RD53B/work/sim/../../cds/../sim/verification_environment/cmd/sv/cmd_checker.sv(102)
@ 42449697000: uvm_test_top.m_env.m_cmd_analysis_env.m_cmd_checker [CMD_CHECKER_MATCH] WRITE
REGISTER        Chip #31,         no auto-increment, address VCAL_HIGH,    data c8 encoded cmd matches decoded
cmd WRITE REGISTER        Chip #31,         no auto-increment, address VCAL_HIGH,    data c8

UVM_INFO
/home/cesar_gr/RD53B/work/sim/../../cds/../sim/verification_environment/cmd/sv/cmd_checker.sv(102)
@ 42599841000: uvm_test_top.m_env.m_cmd_analysis_env.m_cmd_checker [CMD_CHECKER_MATCH]
```

Over 2000 commands sent to the Command Decoder



Example of Verification Catching Bugs: Command Error Test

- I create a Command Error by sending a command with an invalid frame.
- When the following is sent instead of a Cmd Error raised, the command is sent:
    - [WrReg, ChipId][Data, Data][Data, Data][Data,**Invalid**]



Verification of Full RD53B Chip

- ***What does it take to verify a full chip?***
  - A large number of tests written
  - REMEMBER: Over 119K lines of code written just for verification
- Large number of tests necessary to ensure complete **code coverage**.
- Overall 96% of code covered
  - Excluded Analog (vcd/ams) verification

Name	Block Covered Grade	Toggle Covered Grade	Code Covered
(no filter)	(no filter)	(no filter)	(no filter)
pixel_chip	95.99%	82.49%	70305326 / 82533720 (85...
ChipBottom	83.14%	76.67%	648606 / 819186 (79.18%)
PixelArrayReadout	82.57%	75.59%	535997 / 681400 (78.66%)
TwoFFSyncNoCmdResetB	100%	33.33%	4 / 8 (50%)
RINGOSC_BLOCK_A	90.24%	73.73%	613 / 813 (75.4%)
RINGOSC_BLOCK_B	88.67%	100%	2752 / 2792 (98.57%)
DigitalChipBottom	95.15%	81.73%	106553 / 127824 (83.36%)
ResetPulseCalGenerator	99.15%	96.43%	279 / 286 (97.55%)
SelfTrigger	100%	12.1%	2513 / 3058 (82.18%)
ChannelSync	96.47%	75.67%	327 / 403 (81.14%)
CommandDecoder	98.07%	89.08%	1326 / 1425 (93.05%)
GlobalConfiguration	98.08%	83.44%	6758 / 7913 (85.4%)
TriggerTable	93.51%	88.1%	34904 / 39180 (89.09%)
DataConcentrator	96.05%	83.2%	36594 / 43761 (83.62%)
CdcResetSyncAurora	100%	100%	21 / 21 (100%)
CdcResetSyncDataMerger	100%	100%	21 / 21 (100%)
OutputCdcFifoAurora	100%	99.01%	1225 / 1237 (99.03%)
ServiceData	81.65%	87.59%	1260 / 1464 (86.07%)
ServiceCdcFifoAurora	100%	92.51%	1004 / 1084 (92.62%)
auroraDesData[0]	n/a	100%	2 / 2 (100%)
auroraDesData[1]	n/a	100%	2 / 2 (100%)
auroraDesData[2]	n/a	100%	2 / 2 (100%)
auroraDesData[3]	n/a	! 0%	0 / 2 (0%)
auroraDesData[4]	n/a	100%	2 / 2 (100%)
auroraDesData[5]	n/a	100%	2 / 2 (100%)
auroraDesMonitor[0]	n/a	100%	2 / 2 (100%)
auroraDesMonitor[1]	n/a	100%	2 / 2 (100%)
auroraDesMonitor[2]	n/a	100%	2 / 2 (100%)
auroraDesMonitor[3]	n/a	! 0%	0 / 2 (0%)
auroraDesMonitor[4]	n/a	100%	2 / 2 (100%)
auroraDesMonitor[5]	n/a	100%	2 / 2 (100%)
DeserAurora	95.33%	64.3%	13168 / 20074 (65.6%)
auroraMergedData	n/a	100%	2 / 2 (100%)
MergerDataArbitrer	100%	86.67%	54 / 58 (93.1%)
auroraMergedMonitor	n/a	100%	2 / 2 (100%)
MergerMonitorArbitrer	100%	86.67%	54 / 58 (93.1%)

Conclusion

- RD53 Collaboration is working on the production read-out chips for ATLAS and CMS
- ATLAS Chip Submission on Jan. 15, CMS a few months later
- Verification is a cornerstone of chip design and fabrication
- Making up 60-70% of the chip design, a comprehensive and flexible verification methodology is necessary
- Universal Verification Methodology and SystemVerilog used to ensure the proper behavior of the chip design
- Extensive simulation and verification program essential for the design of the final chip

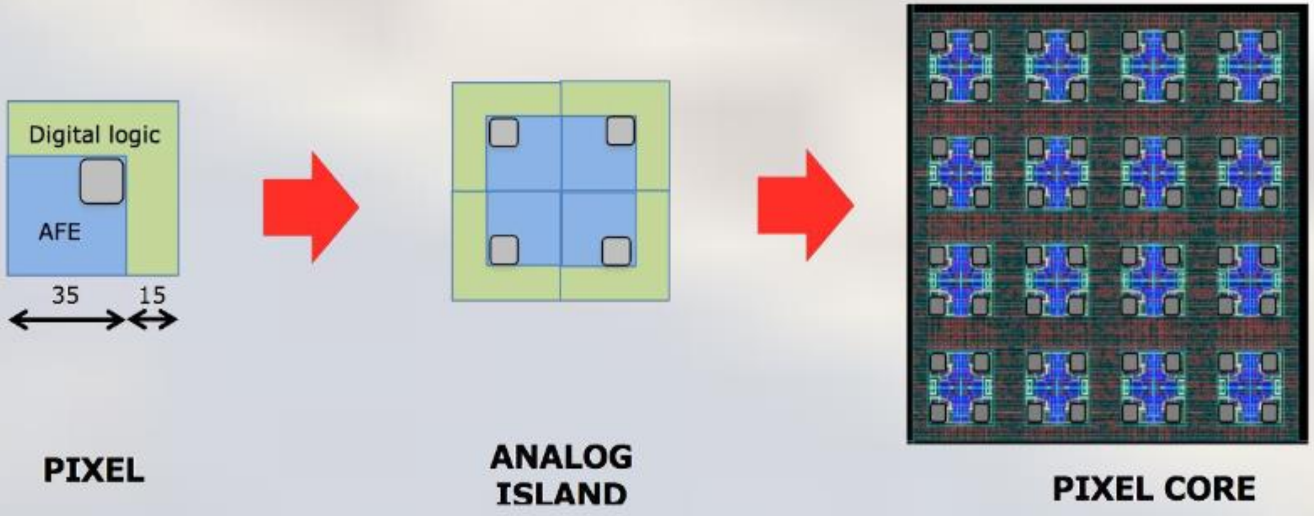
Backup

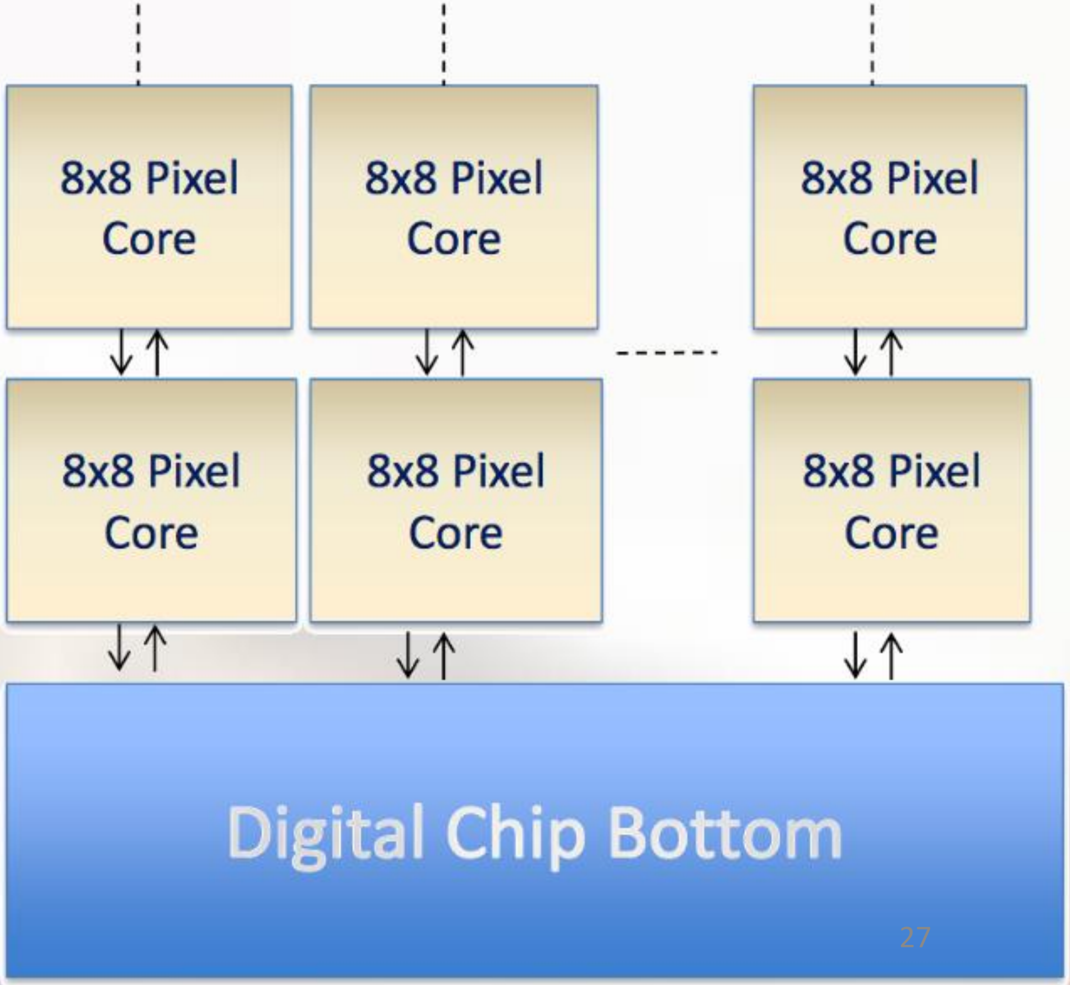
RD53B New Features

- Selected Diff. Front End for ATLAS, Lin. FE for CMS
- Added edge, top, and corner pixel biases
- Upgraded calibration injection & corrected column variation
- New hit synchronization and ToT with 6b-to-4b compression
- Redesign of the startup and generation of reference voltages
- Improvement of SLDO and addition of low power mode
- Addition of overcurrent and overvoltage protection
- Redesigned PLL for lower jitter and robust locking
- Added trigger tags and new readout format with compression
- Added 2-lever trigger for ATLAS and self-trigger
- Added suppression of low charge isolated hit backgrounds
- Added data aggregation between chips
- Changed reset scheme to synchronous and added CMD activity reset
- Extensive triplication and SEU hardening
- Added new resistive temperature sensors and E-fuses for SN
- Added precision ToT and ToA
- Enlarged wire bond pads

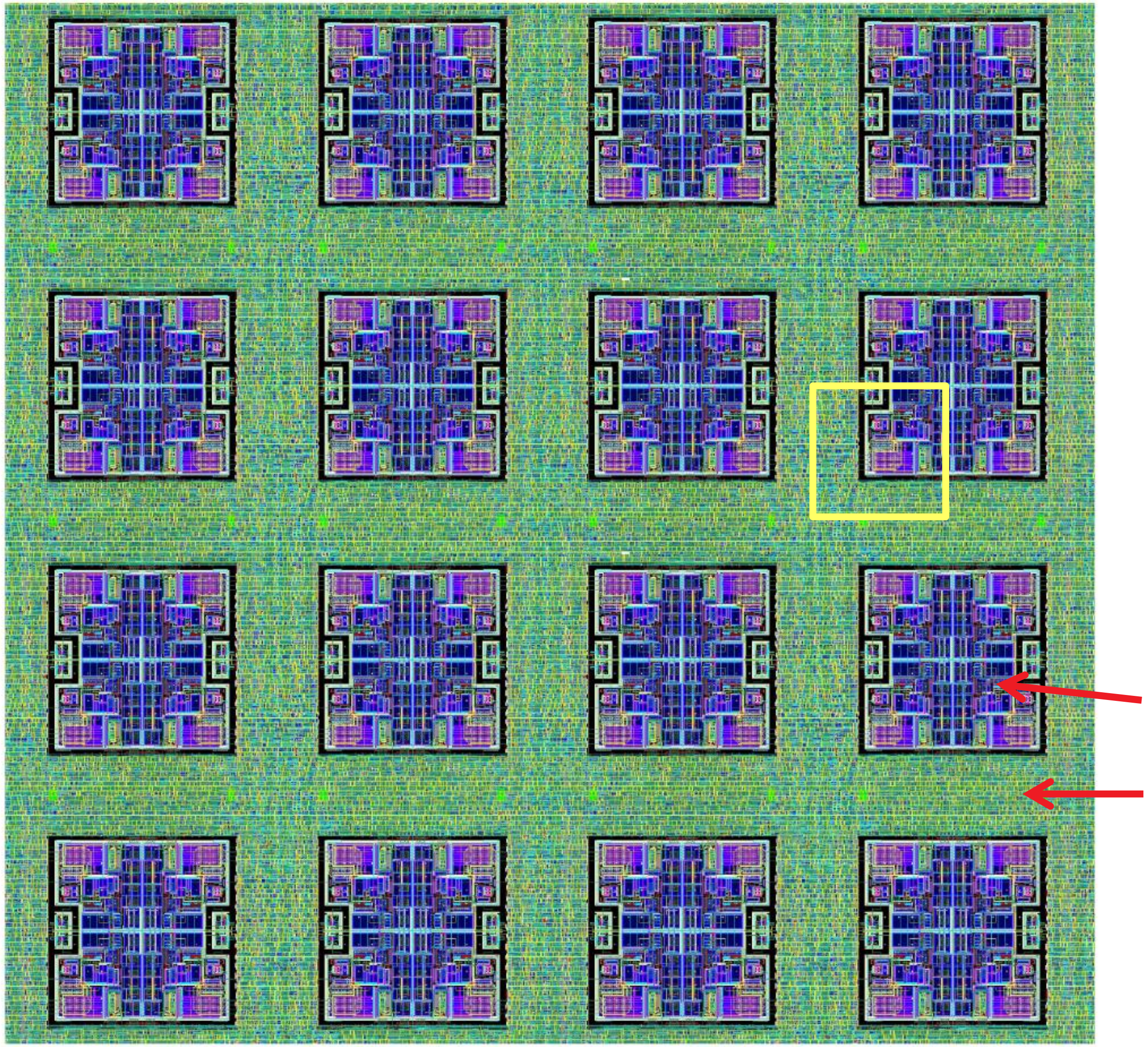
RD53A Pixel Floorplan

- Pixel matrix build from 8 x 8 pixel cores
- One Pixel Core contains multiple Pixel Regions with shared logic and trigger latency buffering
- FEs are placed as analog islands in the "digital sea" of synthesized core logic
- All cores are identical





One Core



One flat synthesized circuit

~ 200k transistors

64 pixels in 16 “analog islands”

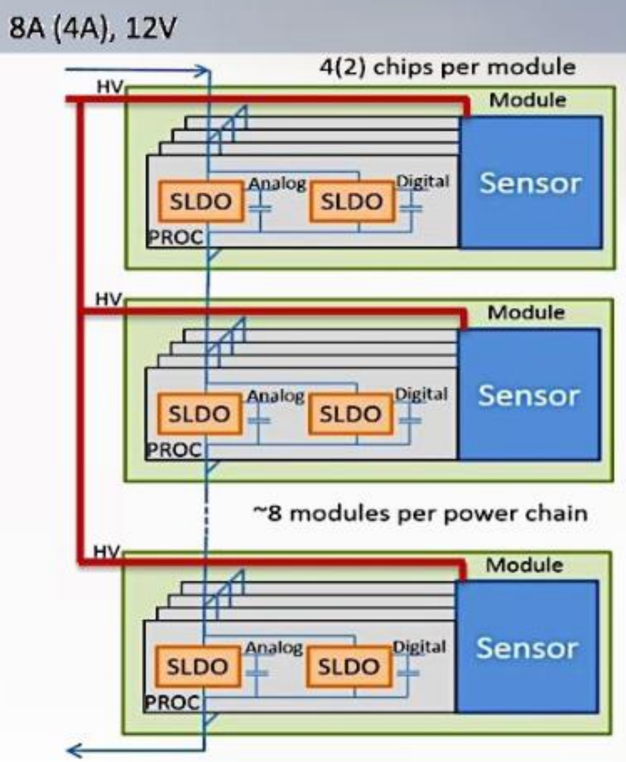
Whole core is stepped
and repeated to make the
pixel matrix

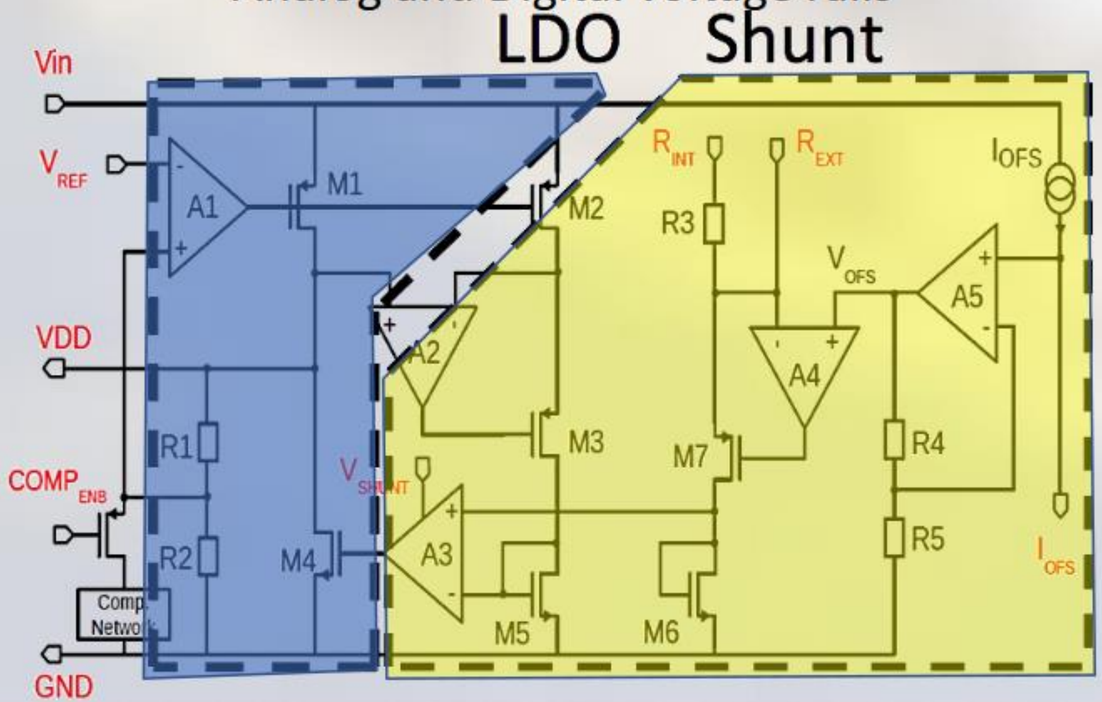
Hand-drawn transistors

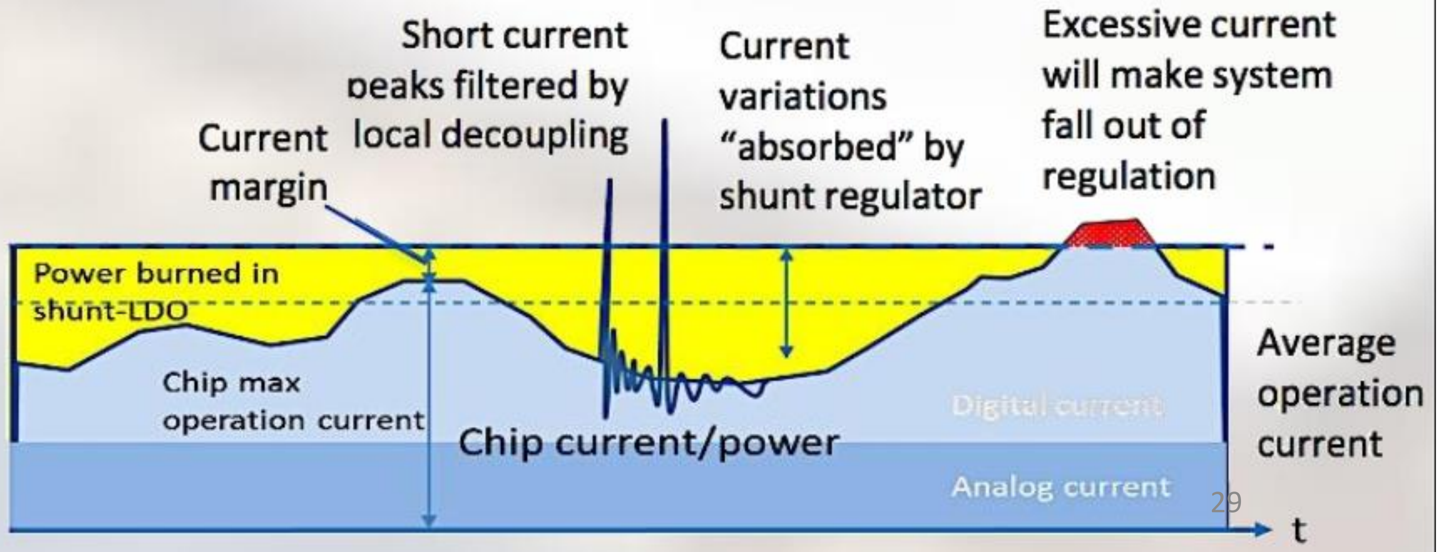
“compiled software”

RD53A: Powering

- **Baseline for powering the ATLAS and CMS HL-LHC pixel detectors**: **serial powering**
  - Low mass, Not sensitive to voltage drop, Low noise
  - Up to 4 chips in parallel resolve single chip failures
- **RD53A Shunt-LDO regulator**
  - Low-drop linear voltage and a shunt regulator
  - Three operational modes (direct, LDO and ShuntLDO)
  - Analog and Digital voltage rails







Output from the Chip(s)

Chip Output Data:

- There are 1 to 4 Aurora64/66b @1.28GB/s encoded lanes that form a Simplex Aurora Channel. The Data has strict alignment. [always Same data type on all Lanes]
- Optionally it has one or two 320Mb/s (640Mb/s, if working) Aurora64/66b encoded Lanes that form a Simplex Aurora Channel only used for on chip data merging.

Aurora Channel contents:

Each Aurora Channel might contain **Pixel Data**, **Service Data** or **No Data**.

- **Pixel Data**: Continuous stream of 66 bit Aurora Frames, without separators. Each frame is identified by a '`01`' header followed by 64 bit of data.
- **Service Data**: Single Aurora Frame identified by a '`10`' header followed by 64 bit of data.
- **No Data**: Standard Aurora Idle Frame, also identified by a '`10`' header.

Service Data:

- A single Aurora Frames that contains the value of **two RD53 registers** (Global or Pixel).

Pixel Data:

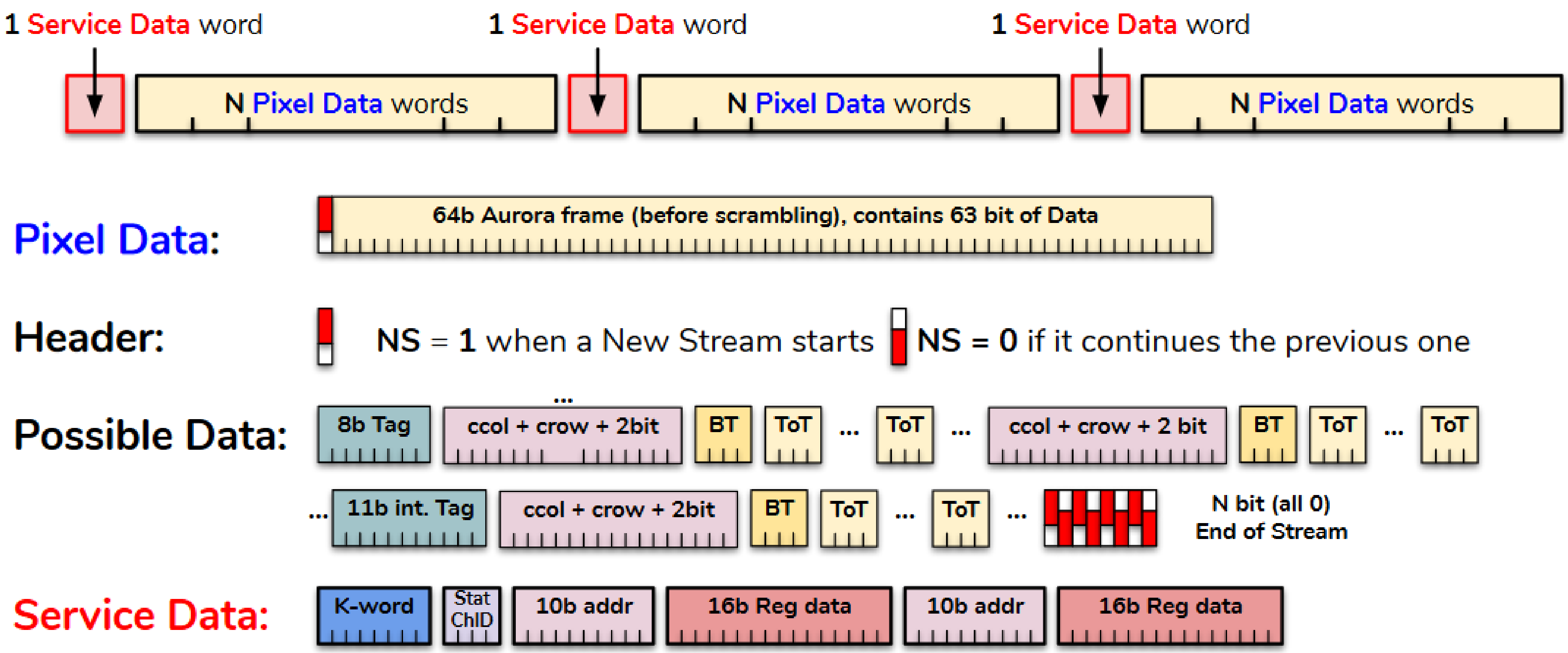
- All data coming from pixels is **split into 64 bit wide Aurora Frames**.
- Data is generated inside the chip and put in variable length streams.
- Each pixel chip always produces an unknown number of concatenated data streams.
- For each incoming Trigger an outgoing Event will be produced [no event suppression]

Output Data [1, 2, 3, 4 links]

The Output Data Stream is sent on 1, 2, 3 or 4 different Aurora physical Lanes.
Each link is logically split in two separate channels:

Pixel Data and **Service Data** where the ratio between the two is user programmable.

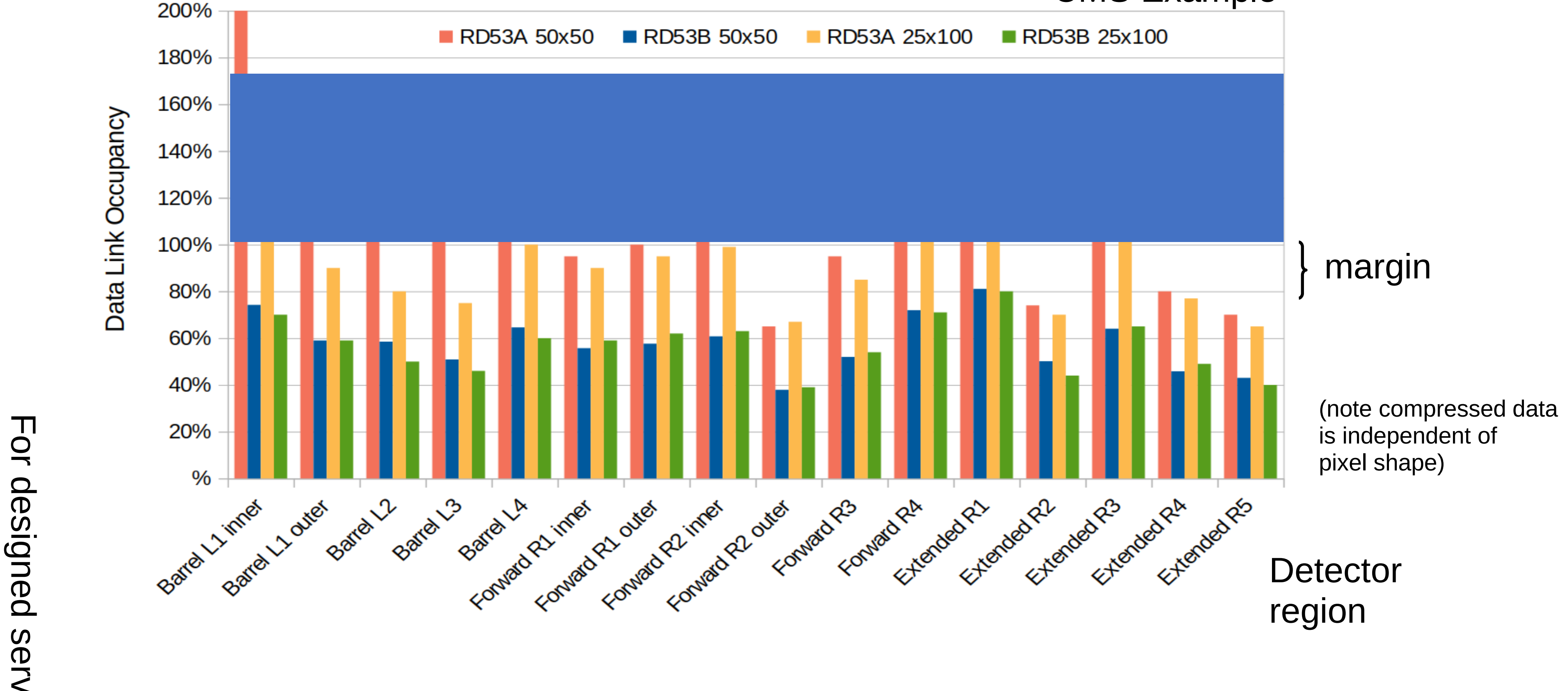
1. Ratio **R** between Pixel Data and Service Data is user programmable.
2. As an example a ratio R = (1 over 50) allows for ~100Mb/s of Service Data



New Feature Highlight

- Encoding with lossless compression used to send data off chip.
  - Custom serial stream encoding to achieve compression AND be tolerant of corrupted fragments
- Important because, thanks to serial power, services volume (and mass) is dominated by data cables.

CMS Example



```
module CommandDecoder (
    //
    // Outputs
    output logic        Trigger,            // Trigger signal. Uses TriggerTag.
    output logic  [7:0] TriggerTag,         // The 8 bit tag associated to each Trigger signal
    output logic        ReadTrigger,        // Signal used to readout data from the chip.
    output logic  [7:0] ReadTriggerTag,     // The 8 bit tag associated data to ReadTrigger
    output logic        Clear,              // Clear the whole data path
    output logic        RdReg,              // [Only if ChipId matches]
    output logic        WrReg,              // [Only if ChipId matches]
    output logic        WrRegMode,          // 0 if Single WrReg command, 1 if continuous mode
    //
    output logic  [8:0] RegAddr,            // Address of the register
    output logic [15:0] RegData,            // Data to be written in the addressed register
    output logic [15:0] BCIDCnt,            // synopsys keep_signal_name "BCIDCnt"          // BCID counter value
    output logic [15:0] TrigCnt,            // synopsys keep_signal_name "TrigCnt"          // Trigger counter value
    output logic [15:0] ReadTrigCnt,        // synopsys keep_signal_name "ReadTrigCnt"      // ReadTrigger counter value
    output logic [15:0] ErrCnt,             // synopsys keep_signal_name "ErrCnt"           // Error counter value
    output logic        CmdErr,             // There is an Error
    output logic [15:0] BitFlipWngCnt,      // synopsys keep_signal_name "BitFlipWngCnt"    // BitFlip Warning  counter value
    output logic        BitFlipWng,         // There is a BitFlip warning
    output logic [15:0] BitFlipErrCnt,      // synopsys keep_signal_name "BitFlipErrCnt"    // BitFlip Error counter value
    output logic        BitFlipErr,         // There is a BitFlip error
    output logic [15:0] SkippedTriggerCnt,  // synopsys keep_signal_name "SkippedTriggerCnt"  // Skipped Trigger counter value
    output logic        SkippedTriggerCntWng, // There have been skipped triggers  . . . . . . [to ServiceData]
    //
    // GlobalPulse
    output logic        GlobalPulse,        // Signal used to generate all GlobalConfiguration pulses
```

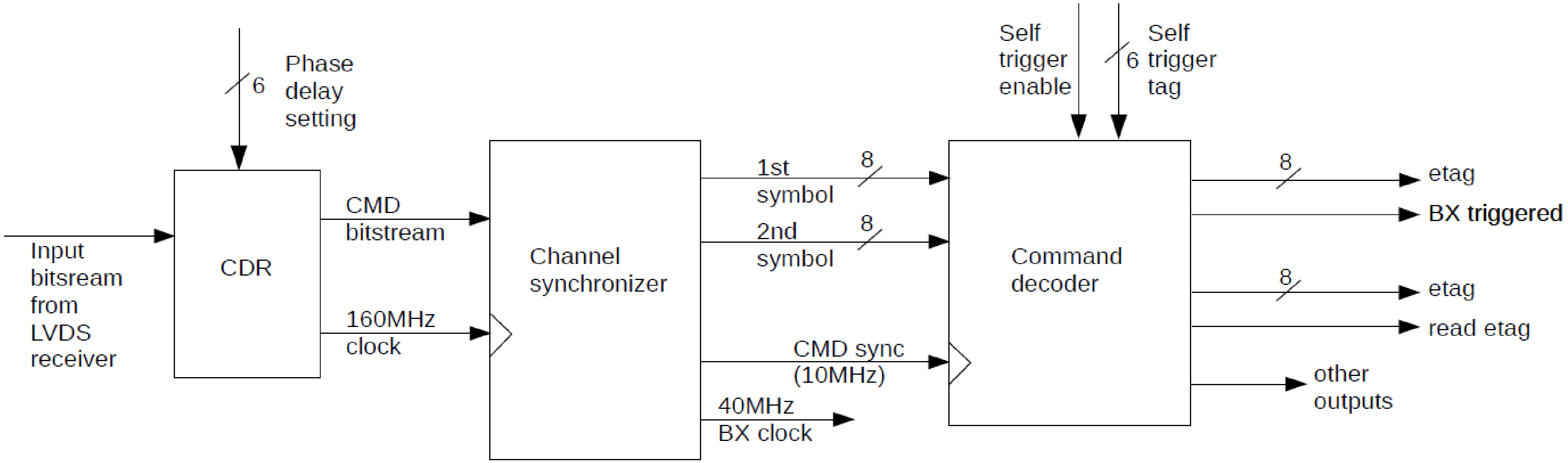
devel | RD53B / rtl / top / | +

Checked how the Clear command is processed inside the chip. Now everyth
Roberto Beccherle authored 5 days ago

Name	Last commit	Last update
..		
ChipBottom.sv	Checked how the Clear command is processed inside the chi...	5 days ago
CoreColumn.sv	Use VCAL* according to V* naming convention	8 months ago
PixelArray.sv	BUG: fixed hit-or endianess mismatches across the design hie...	4 weeks ago
RD53B.sv	FIX: Now if the ChipId changes while a Slow command is exec...	6 days ago

Channel Synchronizer

- In order to lock to a frame alignment we use a Channel Synchronizer
- It receives a "sync" command from the input bitstream
- C.S. then lines up with the start of the frame (align itself to frame boundaries).
- Sync command sent multiple times to ensure locking of frame alignment.
- Once locked, C.S. starts command transmission to the Command Decoder



Verification approach

1. Constrained-random tests

- Random inputs (e.g. hits and triggers at operating conditions, from physics Monte Carlo simulations, input commands with random errors...)
- Automated pixel array verification
- Semi-automated command, configuration and aurora verification
  → increase automation: e.g. configuration reference model, self-correction of errors as in the protocol

2. Directed tests

- Specific test cases aimed to verify single functions and specific test cases not covered 1.
  - mostly used for custom command sequences, injections, etc.
- Quite a few tests checked by looking at waveforms for RD53A. Aim to automate (assertions, ref.model, etc.

3. Analog/digital interface

- Generation of **stimuli for full-spice analog simulations** (dump *vcd* for analog designers)
- Others: mixed signal simulations in analog and/or dedicated tests in digital (AMS model + Real Number Model planned for analog blocks of Analog Chip Bottom, AFEs RNM considered)

Interfacing UVM to the DUT

- SystemVerilog interfaces between UVM components and DUT
  - defined in the top level verilog module (we call it *pixel_chip_harness*)
  - linked to the UVM world by ***set***ting them in the ***uvm_config_db*** (in *pixel_chip_tb.sv*)
- Main interfaces and transactions defined:

Hit interface	**Trigger interface**	**Cmd interface**	**Output data interface aurora**	**Analysis interface** (optional)
• Analog input hit signal	• Input trigger signal	• Input commands	• Output of the Aurora receiver	• Virtual flag signals related to DUT status for statistical information collection (dead time, buffer occupancy, …)

hit_trans

```
+time_ref: int
+hits: hit []
```

hit struct:

```
+charge: unsigned int
+delay: unsigned int
+col: unsigned int
+row: unsigned int
```

trigger_trans

```
+time_ref: int
```

cmd_trans

```
+cmd_type: cmd_type_t
…
+chip_ID: [3:0] bit
+reg_addr: GCR_addr_t
…
```

Transaction with many fields to cover all command types (including configuration commands)

hit_trans

```
+time_ref: int
+hits: hit []
```

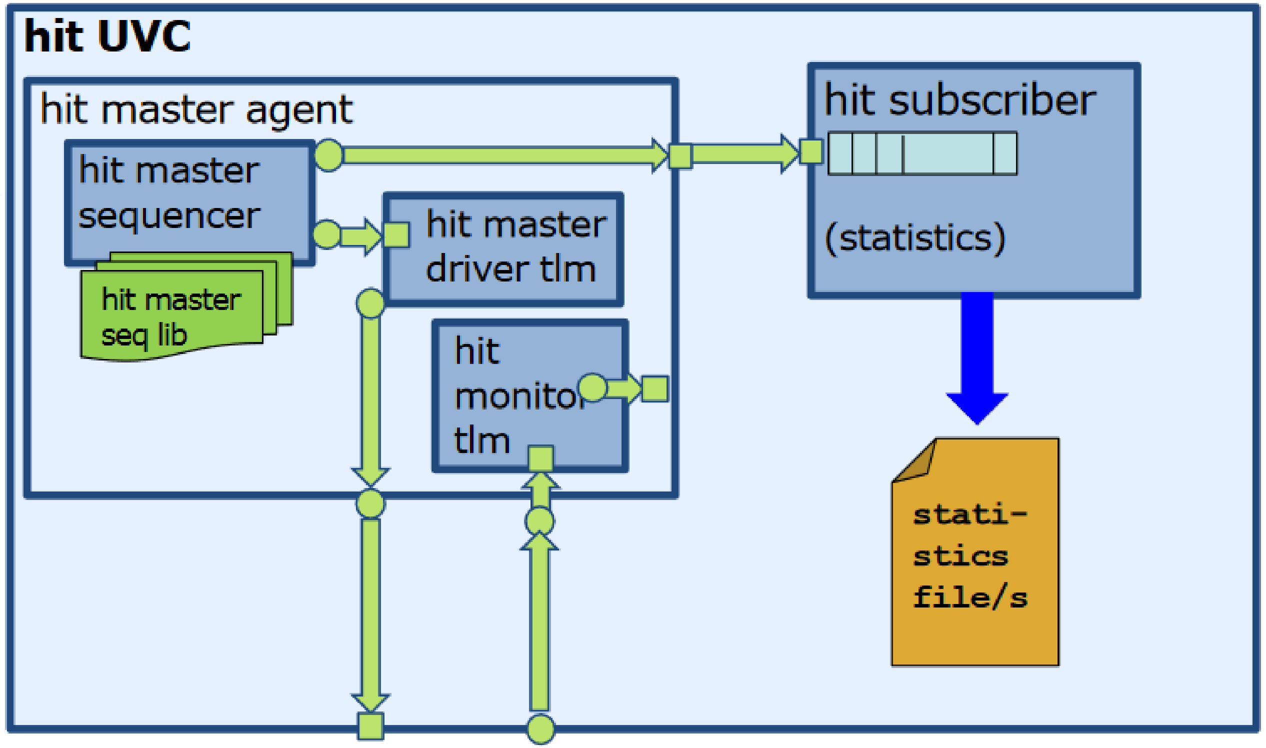
The aurora monitor builds a hit trans for the pixel array scoreboard

analysis_trans

```
...
DUT-specific data members
(e.g. pixel dead-time,
buffer full)
...
```

Example interface UVC: hit

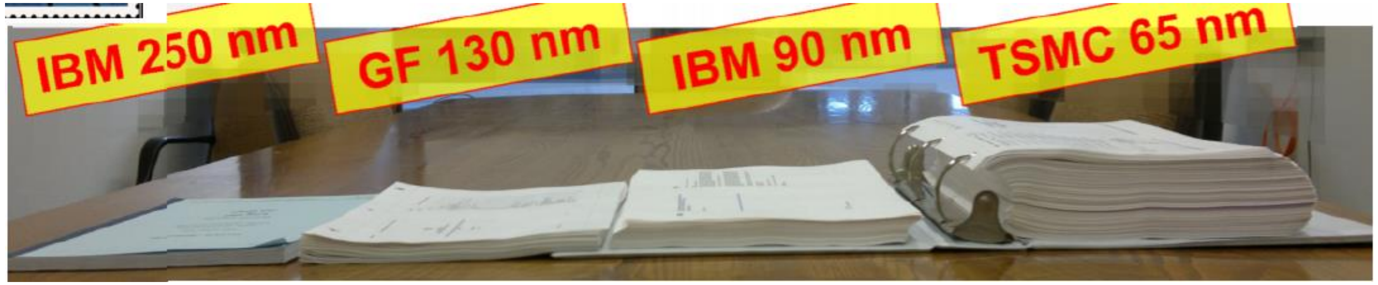
- **Hit generation of input stimulus** handled by:
  - hit master **sequencer** (structural component that runs a sequence)
  - **sequence** (of transactions)
  - **driver** drives to the hit interface
- **Hit monitoring**
  - handled by the hit **monitor** in the agent (the latter is just a wrapper)
- The basic **objects** being exchanged are hit **transactions**

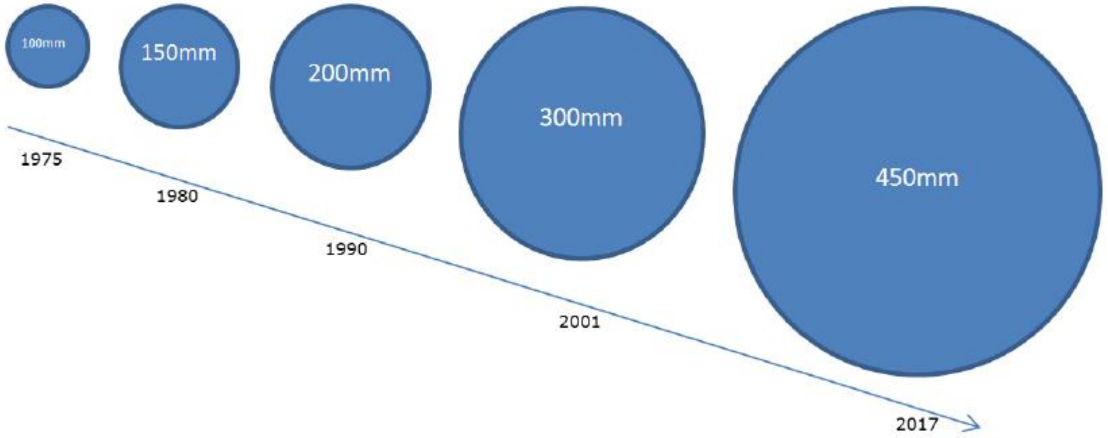


- **Statistic collection:**
  - Hit **subscriber** responsible of dumping/analyzing generated hits (histograms, hit rate computation,..)
- **Configurability through the tests and configuration database**
- *Note*: blocks *_tlm* created for DUT @TLM description: for RTL, additional driver/monitor with *_sig2tlm*, *_tlm2sig* (conversion from/to *signal* level)

Why Verification is so Important?

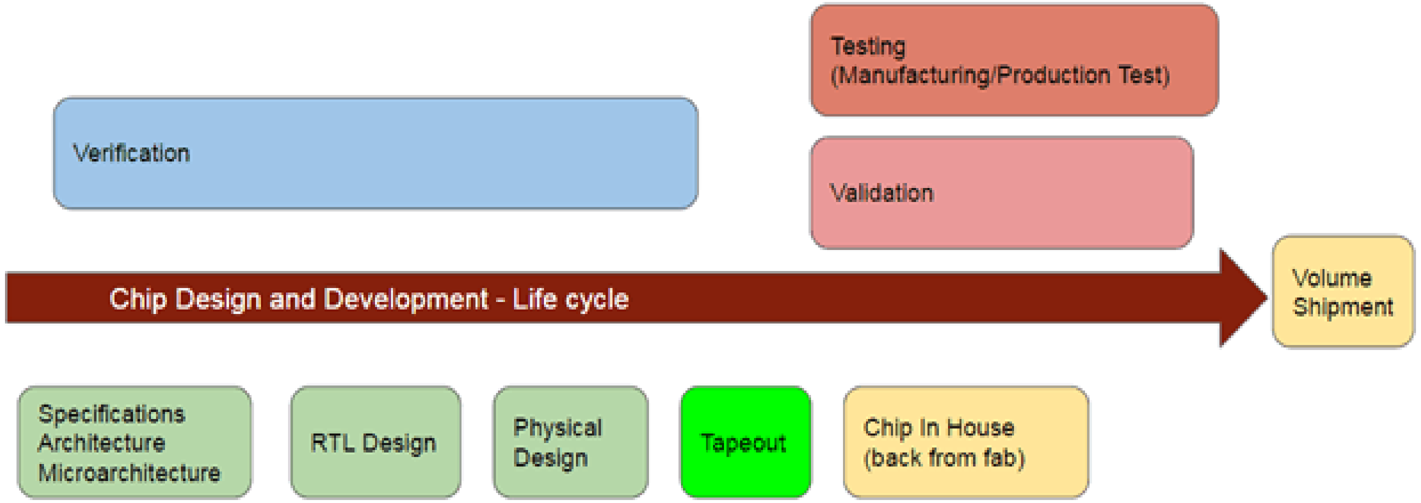
- Increasing complexity of IC design means much higher chance of errors and bugs present in final product.
- Verification ensures (to an extent) the design is bug free and behaves as expected.
- This results in an overall faster design process which avoids major errors that could mean product recalls, time syncs, etc.





What is Verification?

- Ensure the design of the chip is accurate to specification/function/application.
- Basic Steps of Verifying the Design Under Test (DUT):
  - Generate stimulus
  - Apply stimulus to the DUT
  - Capture the response
  - Check correctness of response
  - Measure progress against overall verification goals
  - Ensure maximal coverage



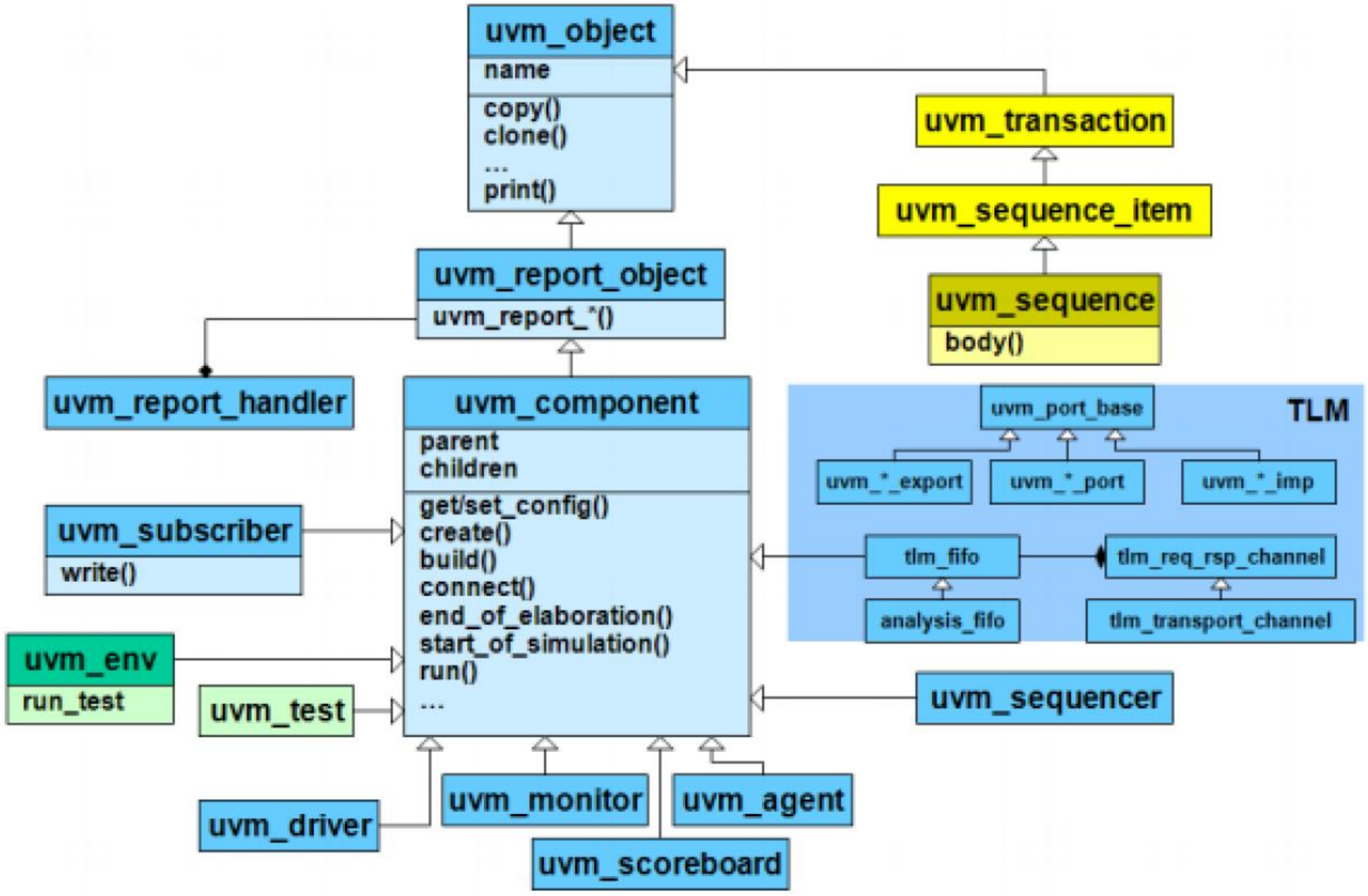
[2]

Universal Verification Methodology (UVM)

- Created in 2011 by Accellera to be used by all three major electronic design automation (EDA) companies: Cadence, Mentor, Synopsis
- Pre-built library and extensive documentation as guidance to reach greatest coverage in verification in a highly efficient manner.
- Contains many useful features:
  - Standard sets of base classes for data and environment
  - Utilities for managing log files and inter-process communication
  - Highly customizable reporting features.
  - Design for reusability

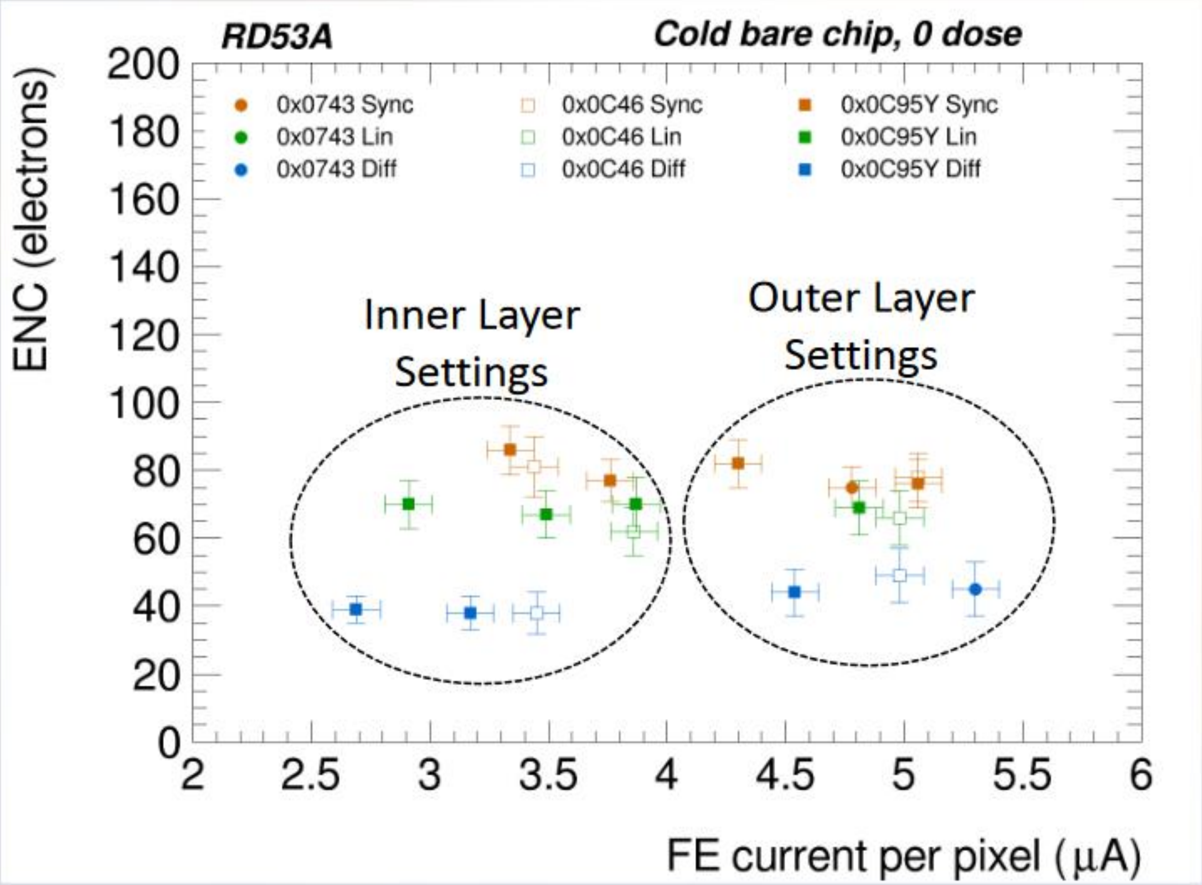
Top-down design of UVM Library

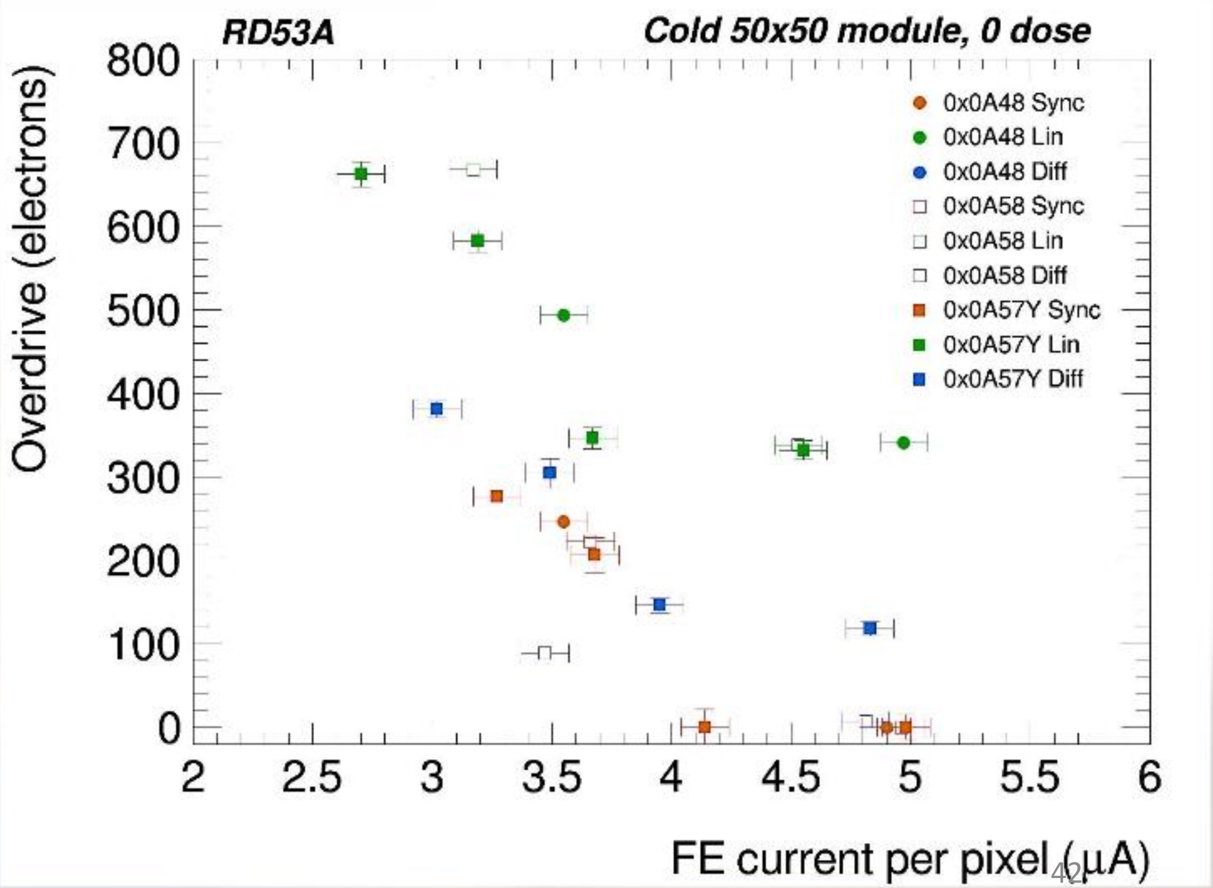
- Classes inherited from **uvm_object**
- **uvm_component:** Used for structural classes
  - Part of the framework for the full simulation run time.
  - Ex: uvm_driver, uvm_monitor, etc.
- **uvm_transaction**: data item or sequence item
  - Multiple instances created and deleted during runtime.



RD53A Results: Analog FE Characterization

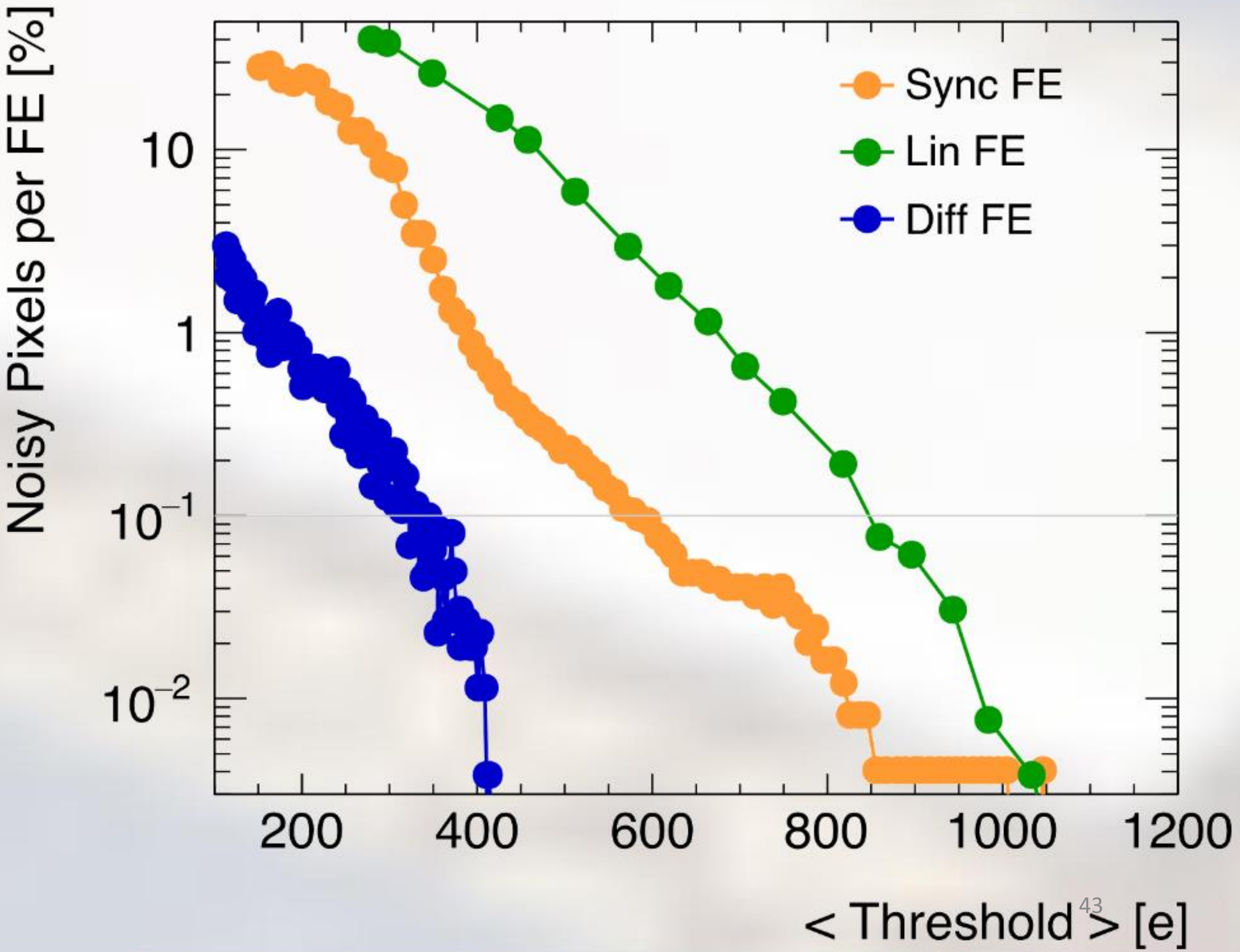
- Verify the functionalities of the three FEs
- Tests with bare chips and 50x50 μm^2 sensors with different settings
- Overdrive: difference between threshold and in-time threshold





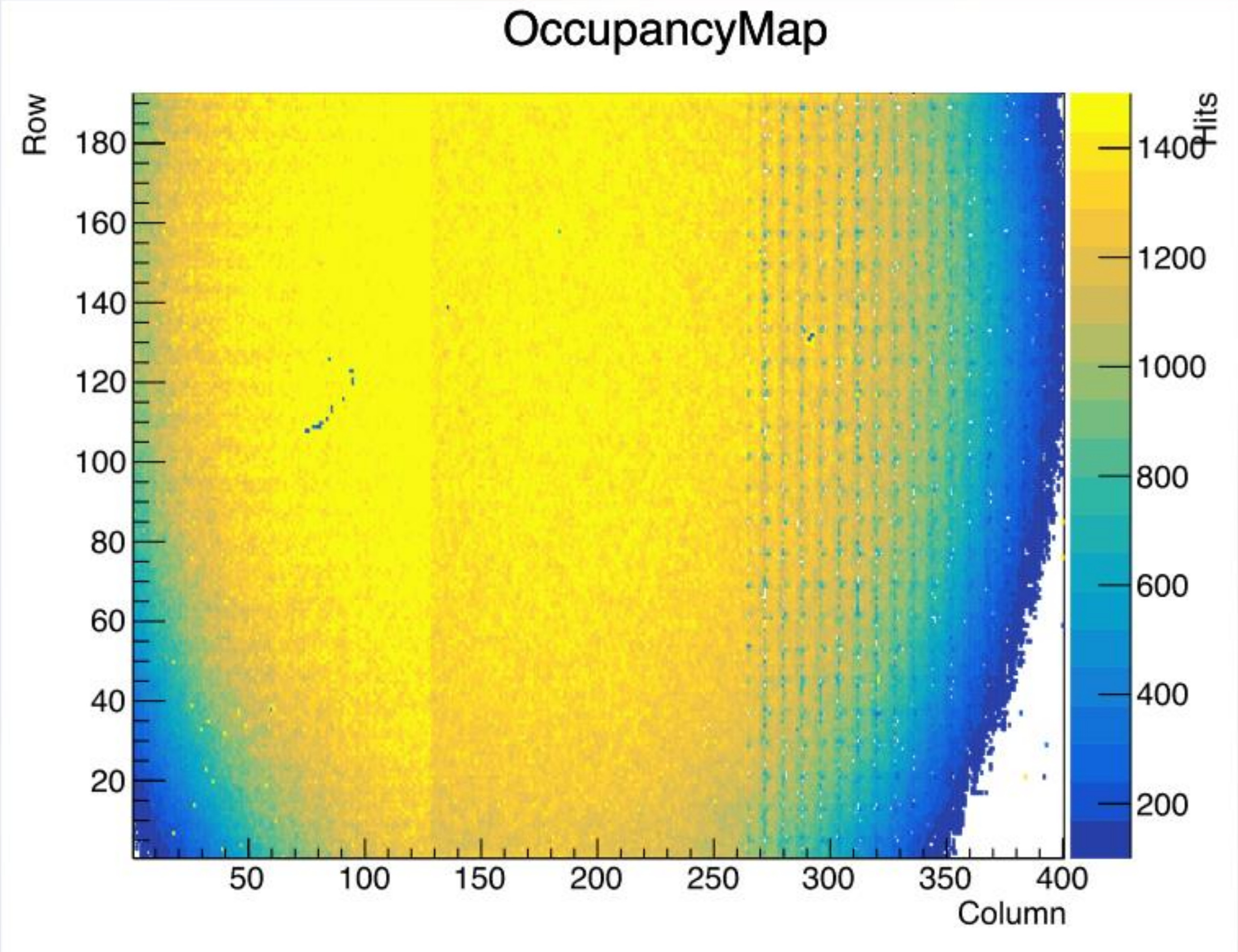
RD53A Results: Analog FE Characterization

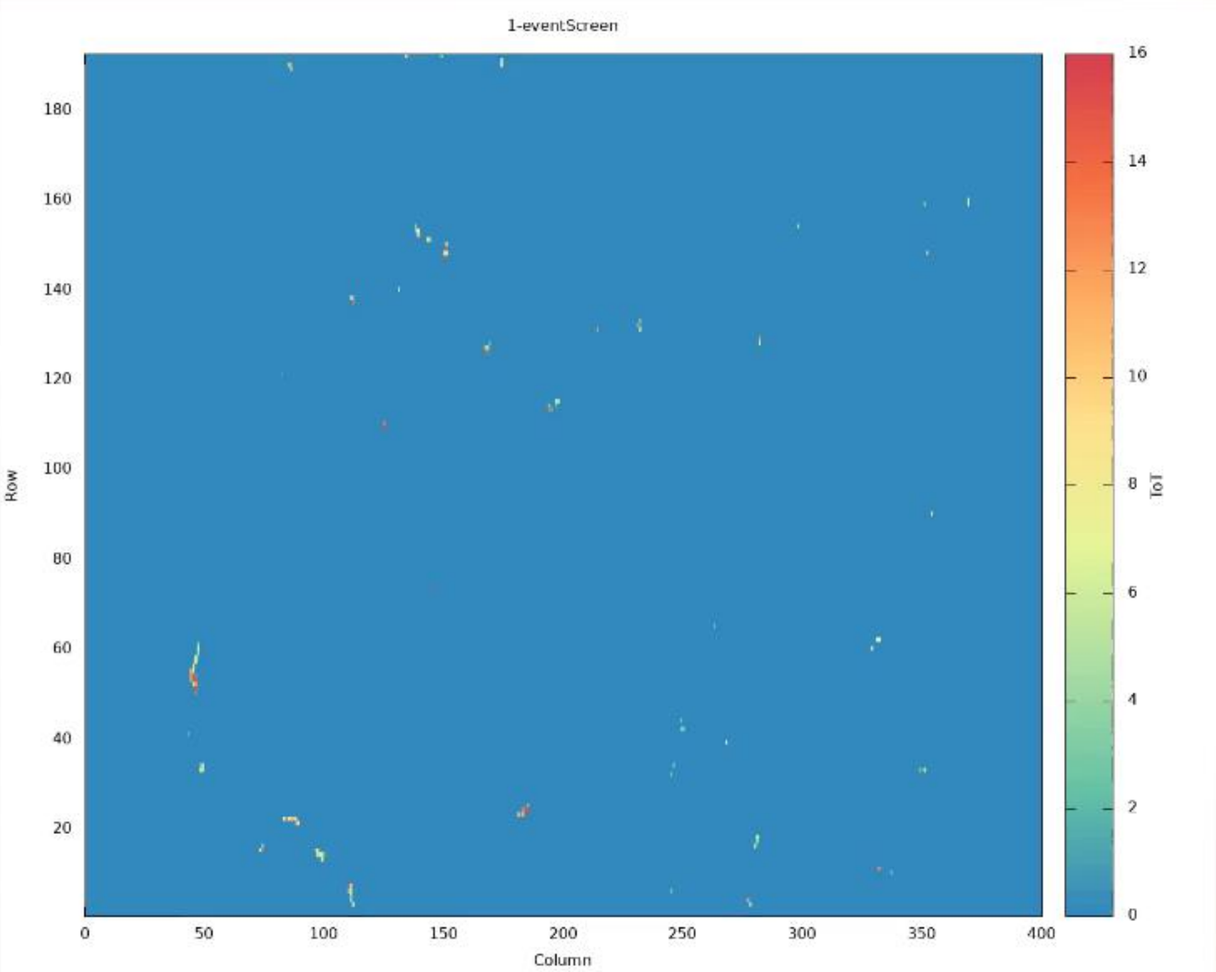
- Measure how many pixels have noise occupancy > 10^{-6} hits/bc as a function of the mean threshold
- Procedure:
  - ✓ tune
  - ✓ lower global threshold
  - ✓ retune pixel threshold (except for the SFE)
  - ✓ measure (in-time)threshold
  - ✓ measure noise
- Dependent on the tuning procedure (assumed that the threshold distributions is Gaussian)
- Cold Bare Chip



RD53A Results: Realistic Hit Rate

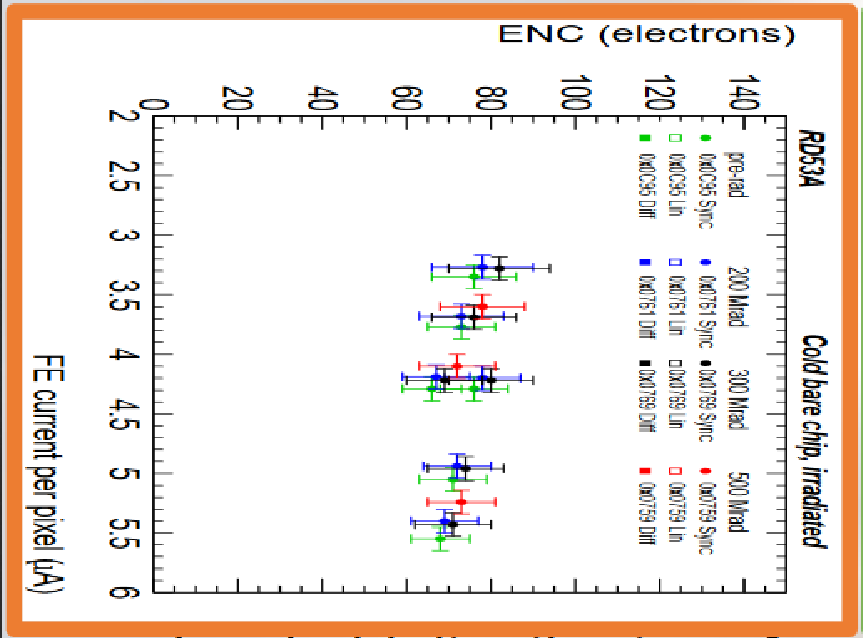
- Kr-85 beta source with activity 60 mCi (opening window 19 mm)
- Estimated hit rate: 7×10^{-4} hits/bc

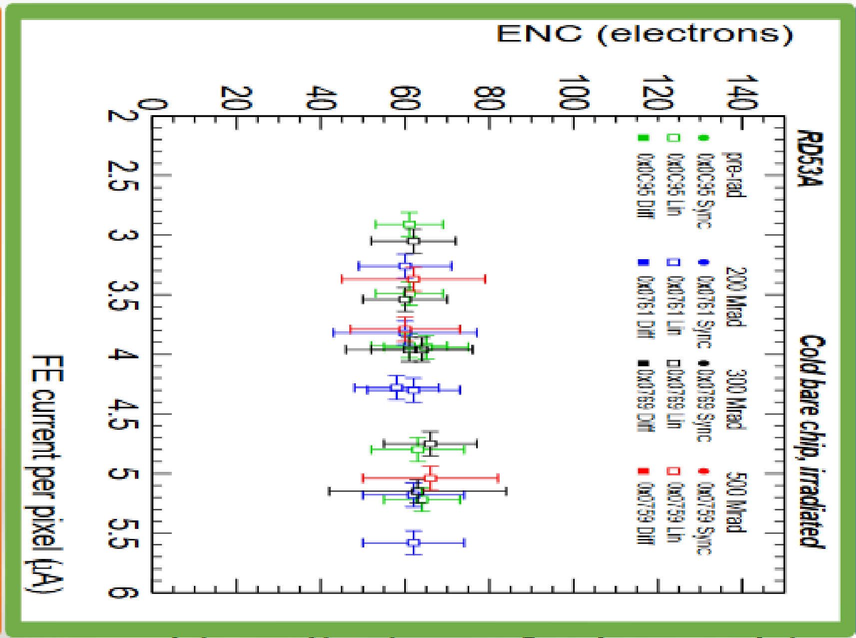


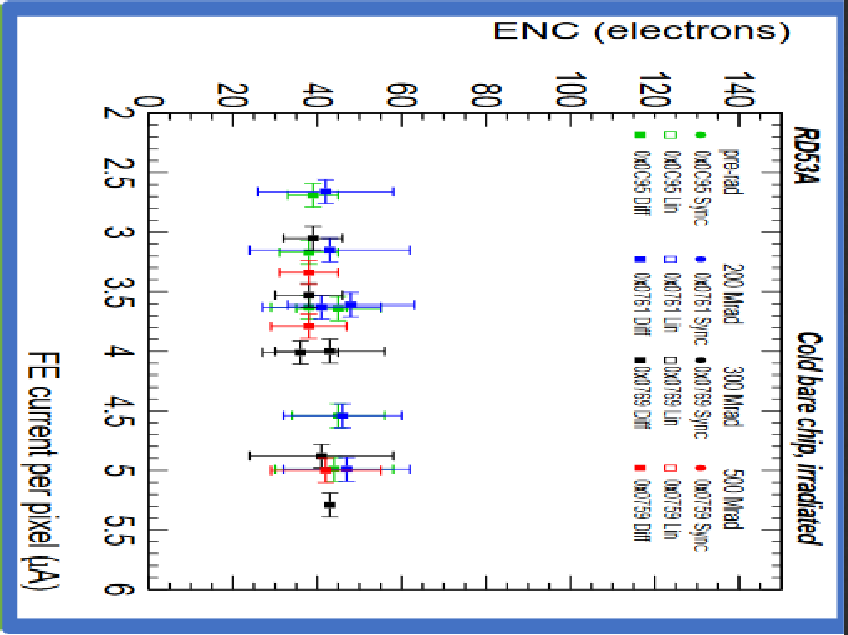


RD53A Results: Analog FE Characterization After Irradiation

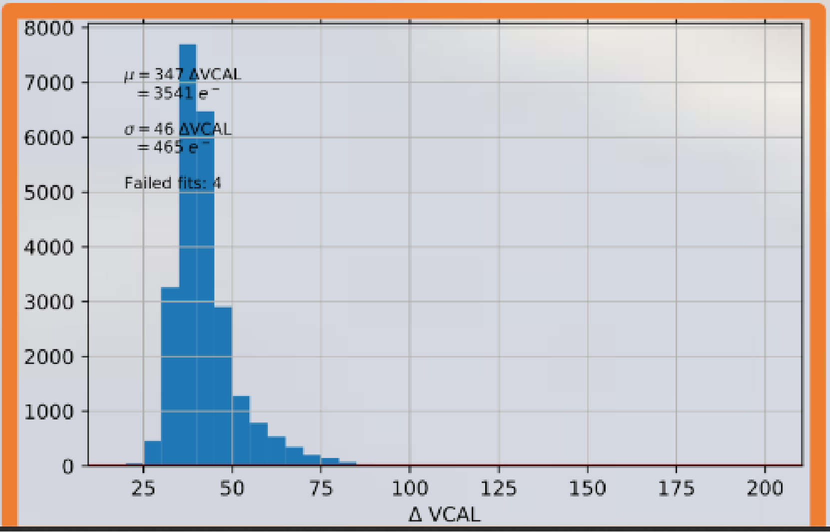
- Noise for different current settings (for 0, 200, 300, 500 Mrad)

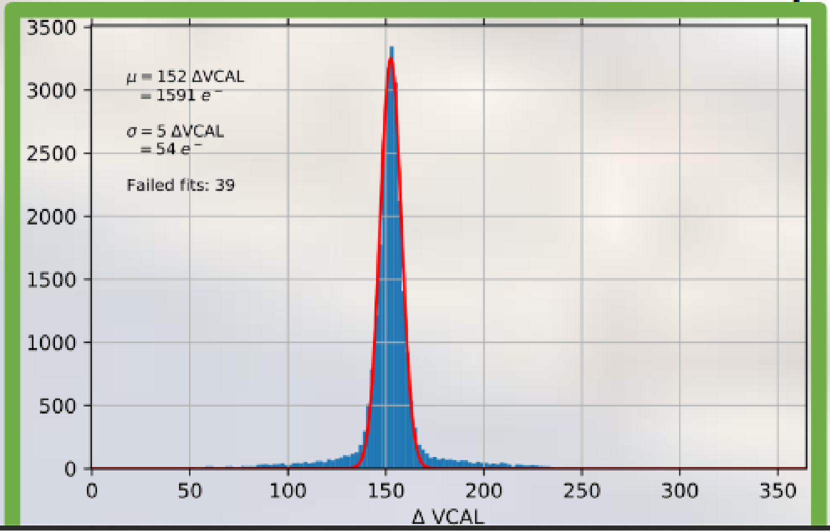


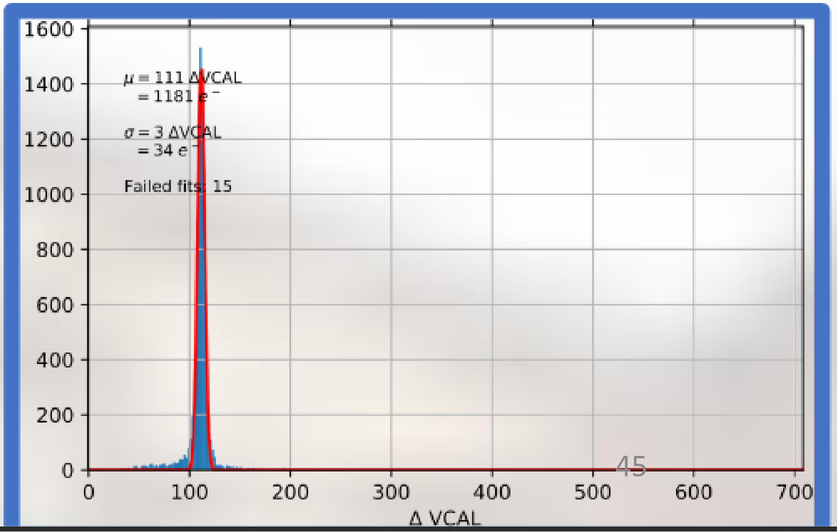




- Threshold distribution after 1 Grad irradiation of a bare chip







References

1. Performance of the CMS Tracker Optical Links and Future Upgrade Using Bandwidth Efficient Digital Modulation - Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/figure/Cross-sectional-view-of-the-CMS-detector-11-The-HCAL-contains-and-measures-the_fig3_41217449 [accessed 3 Dec, 2019]
2. https://anysilicon.com/verification-validation-testing-asicsoc-designs-differences/
3. *Francesconi, Juan Ignacio et al. "UVM based testbench architecture for unit verification." 2014 Argentine Conference on Micro-Nanoelectronics, Technology and Applications (EAMTA) (2014): 89-94.*