ALPIDE Chip - Verification Challenges and Practical Solutions

Svetlomir Hristozkov

14/02/2017

Verification

"Verification is a process used to demonstrate that the intent of a design is preserved in its implementation[1]."

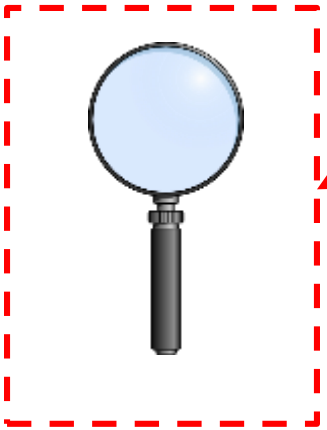
[1] Janick Bergeron (2006), Writing Testbenches Using System Verilog, New York, NY: Springer Science+Business Media

Verification

Ultimately… aims at finding bugs

Verification

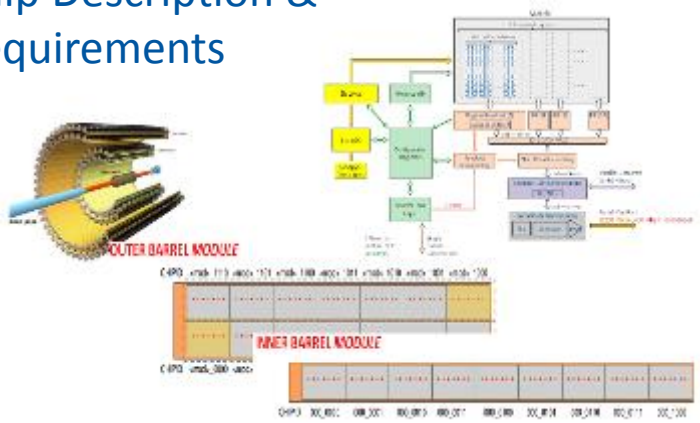
Some tips on how to build ….

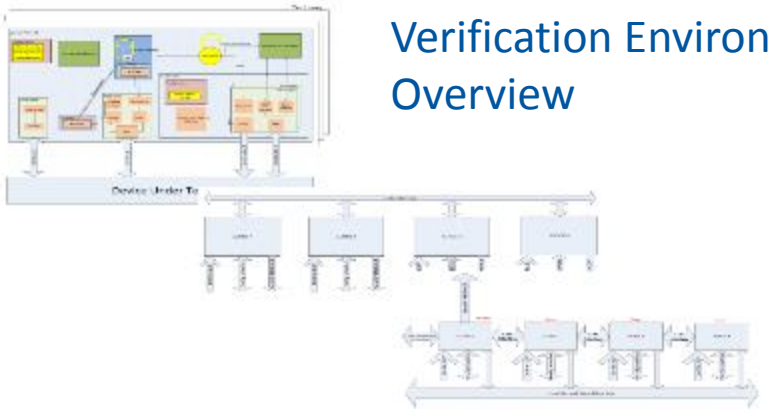


- Verification Environment
- Tools and optimization
- Properties and Formal

Seminar Outline

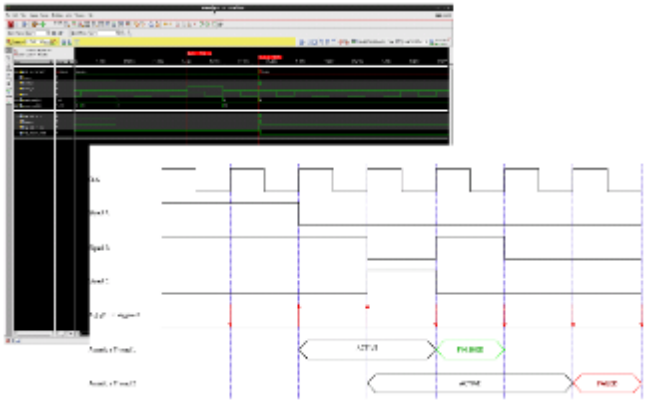
Chip Description & Requirements



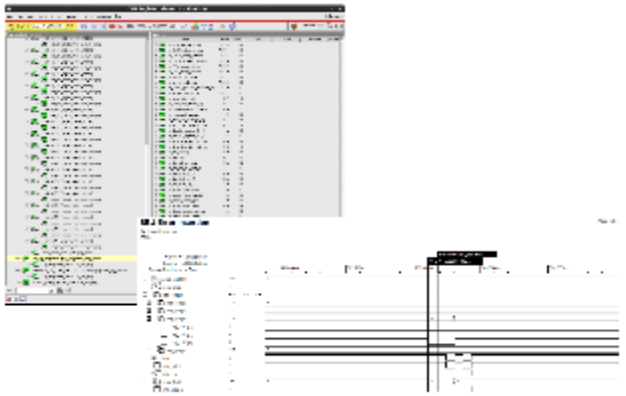


Verification Environment Overview

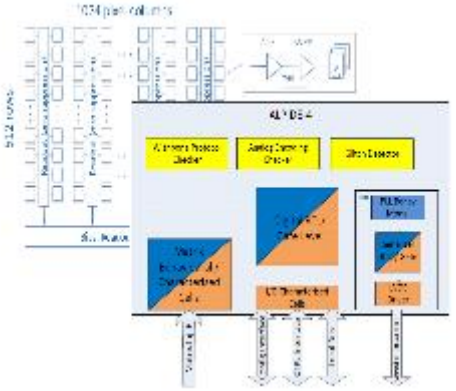
Assertion Based Verification (ABV) and Formal



SEU Protection Verification



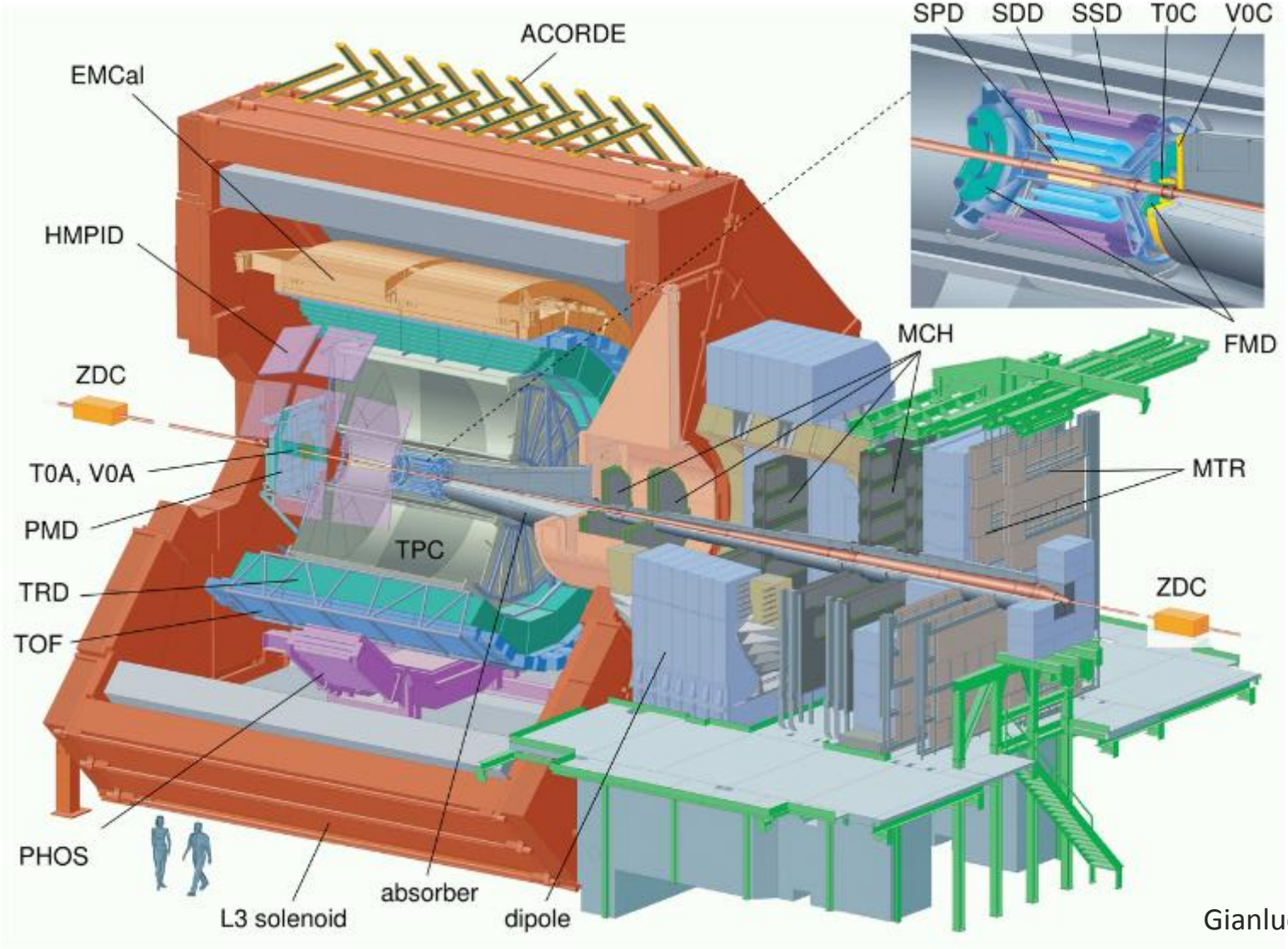
Multi Snapshot Incremental Elaboration



ALPIDE INTRODUCTION

Current ALICE Detector

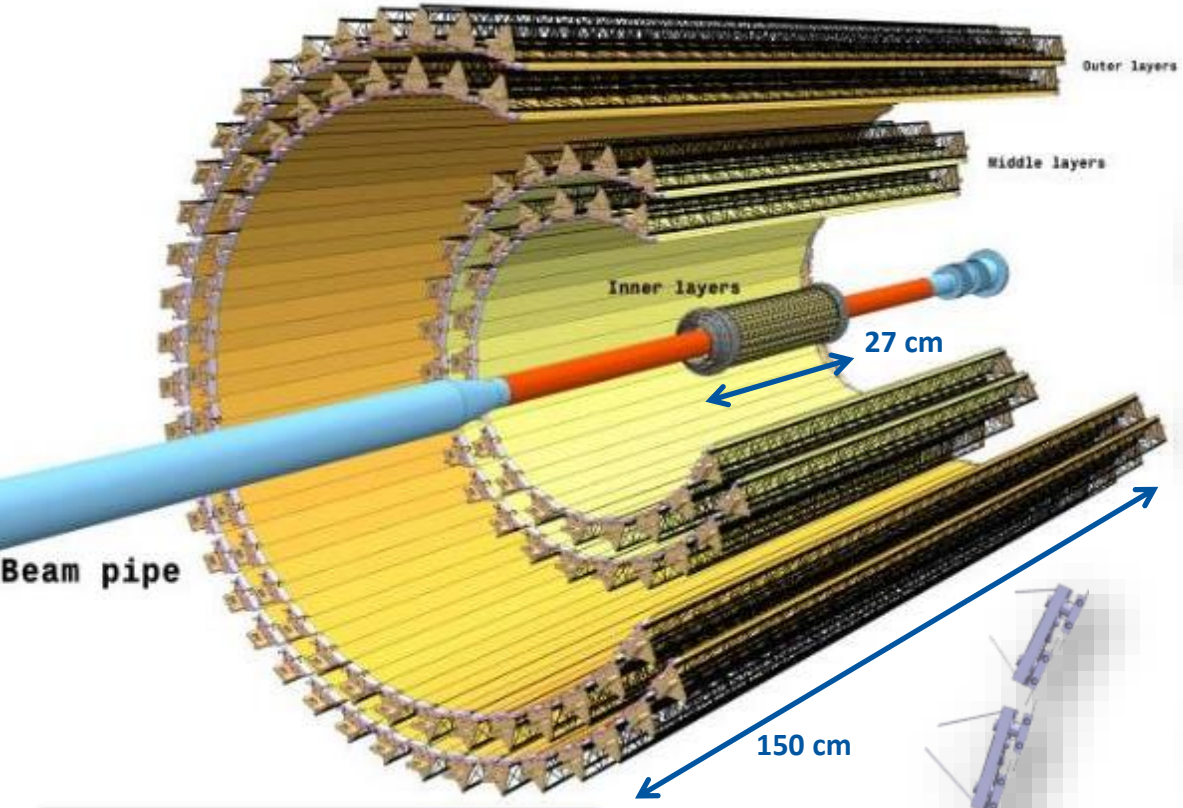
Pixels Drift Strips



Gianluca Aglieri Rinella

Focus: Heavy Ions collisions, Quark Gluon Plasma

New ALICE ITS Layout



10 m² sensitive area
~24000 CMOS Pixel Sensors
12.5 Gpixels

4 Outer Barrel layers (**OB**)
1% X_0

3 Inner Barrel layers (**IB**)
0.3% X_0

40 cm
2.3 cm

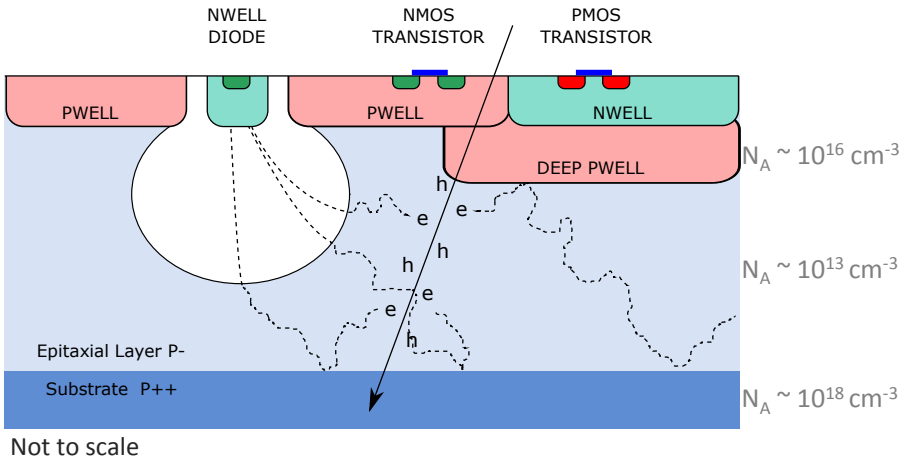
STAVES: 48 42 30 24 20 16 12

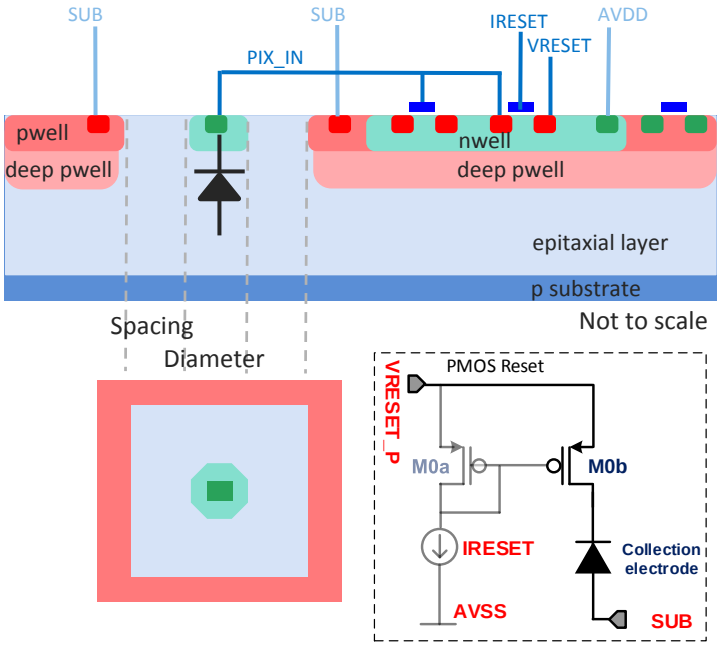
Gianluca Aglieri Rinella

Sensor Technology

Pixel Sensor CMOS 180 nm Imaging Process (TowerJazz)

3 nm thin gate oxide, 6 metal layers

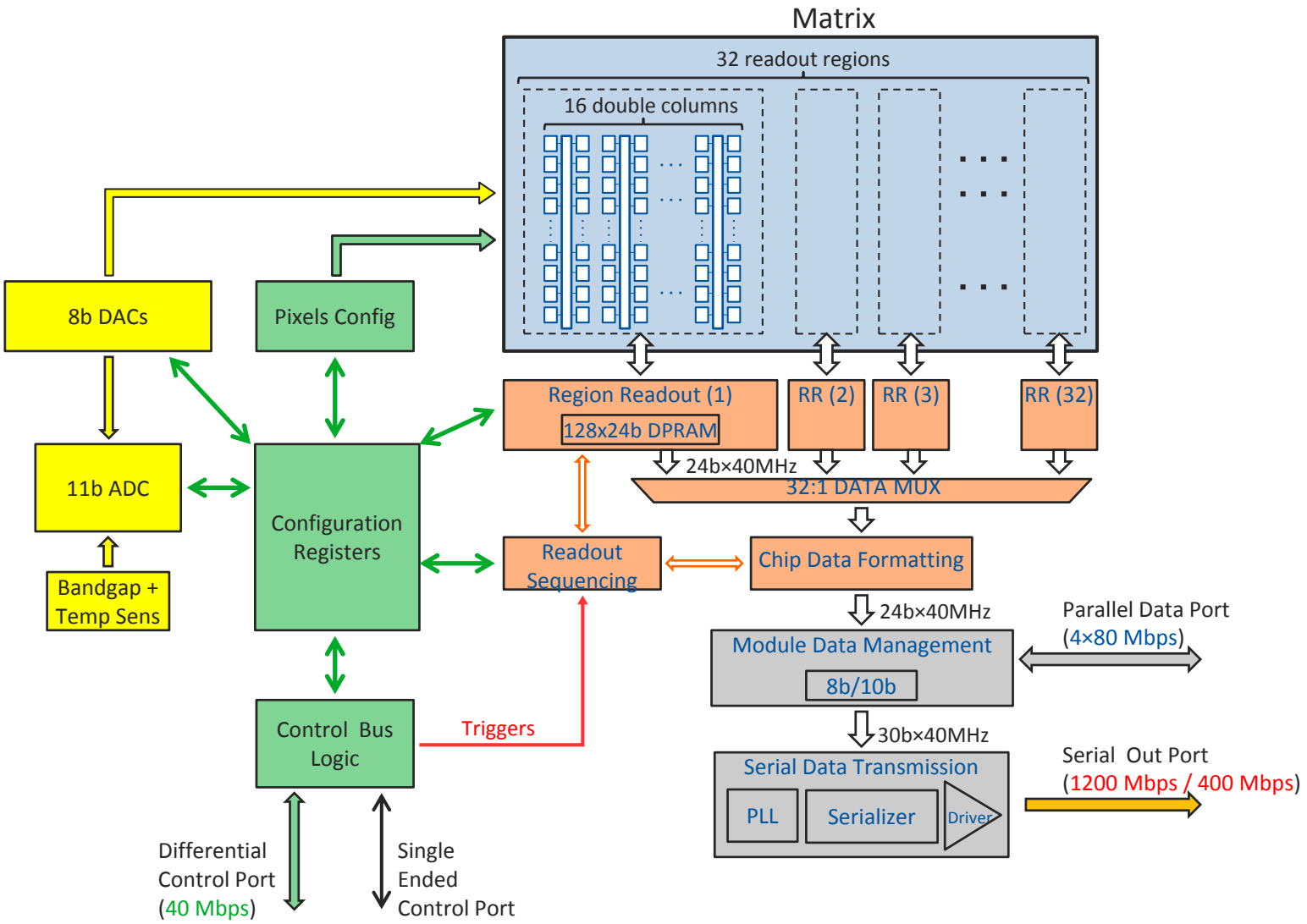




- High-resistivity (> 1kΩ cm) p-type epitaxial layer (18 μm to 30 μm) on p-type substrate
- Deep PWELL shielding NWELL allowing PMOS transistors (full CMOS within active area)
- Small n-well diode (2 μm diameter), ~100 times smaller than pixel => low capacitance => large S/N
- Reverse bias the substrate to increase the depletion volume around the NWELL collection diode

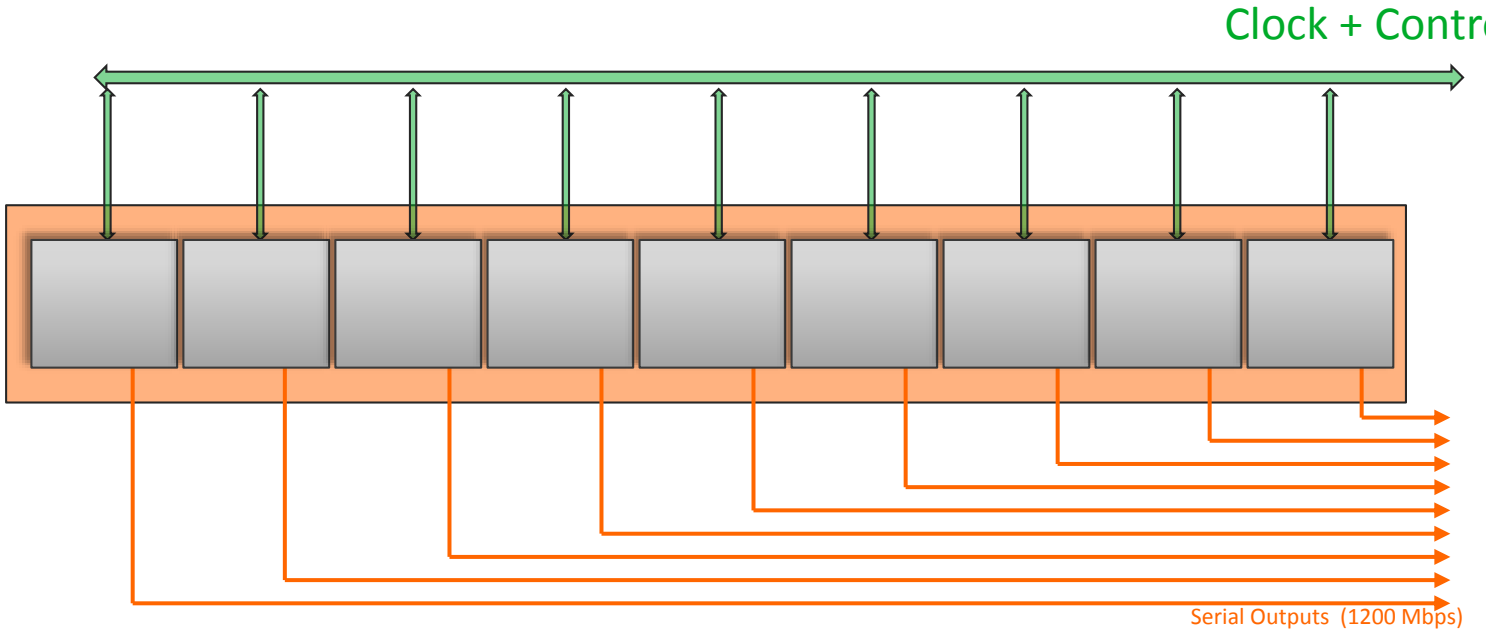
Gianluca Aglieri Rinella

ALPIDE Functional Diagram



Operating Scenarios – Inner Barrel

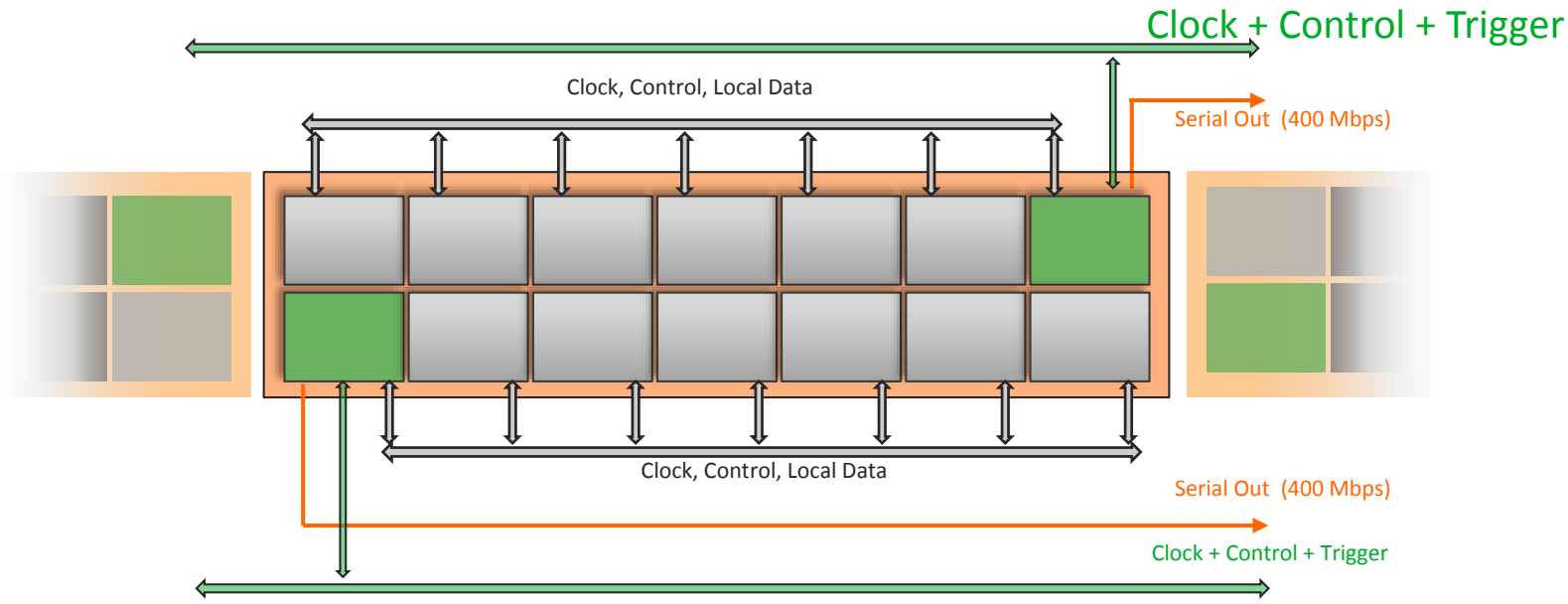
ITS Inner Barrel Module – 9 chips, common clock and control, independent data lines



Gianluca Aglieri Rinella

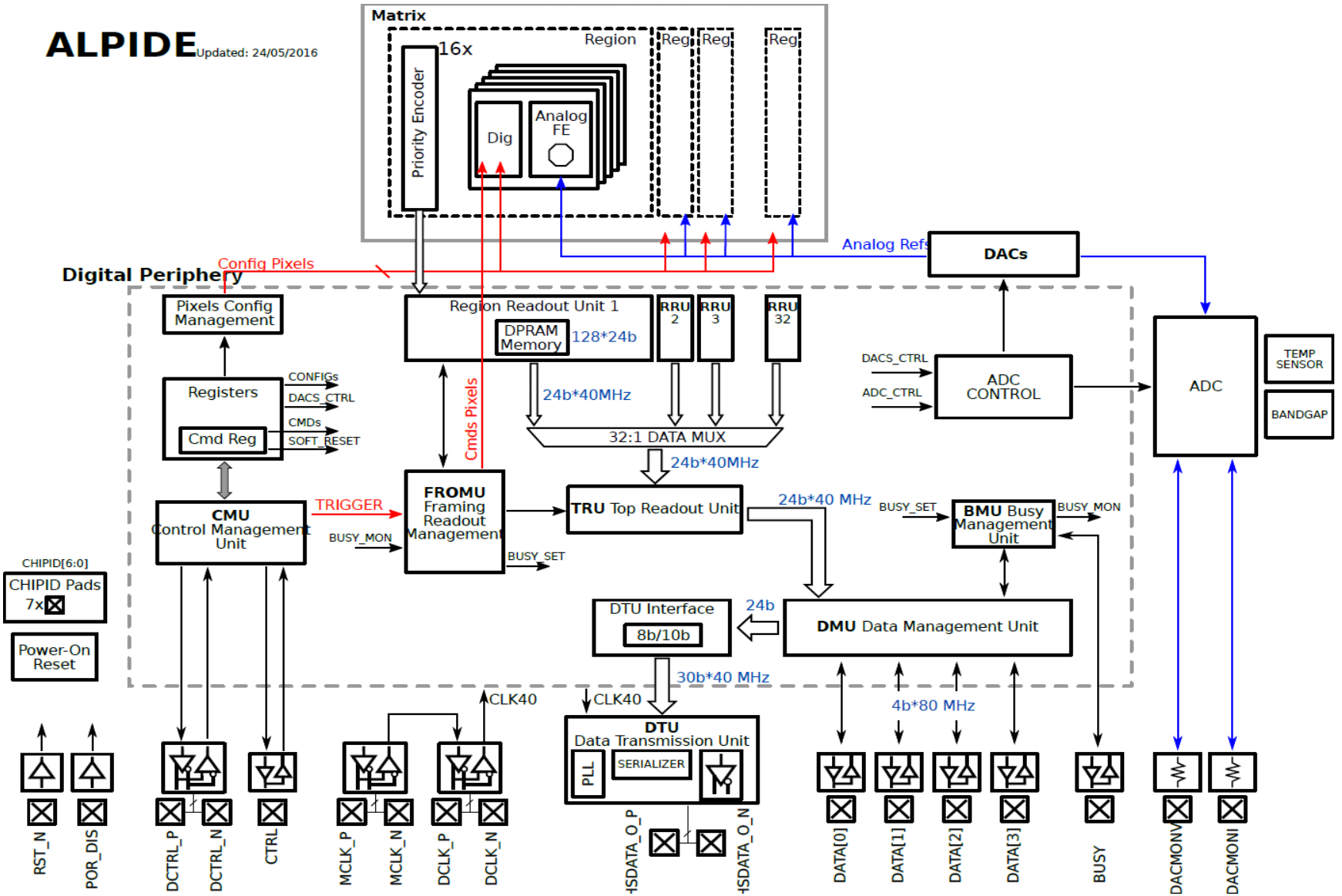
Operating Scenarios – Outer Barrel

ITS Outer Barrel Module – 2 groups of chips, Master + 6 Slaves
- Only the Master interfaces to the external world



Gianluca Aglieri Rinella

ALPIDE Block Diagram



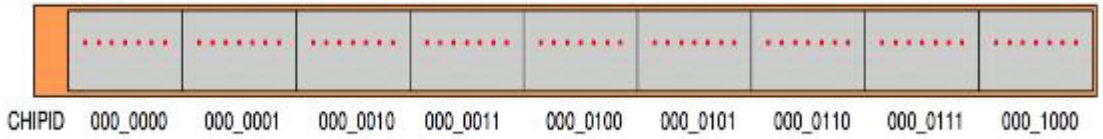
ALPIDE VERIFICATION INTRODUCTION

Design under Verification - Scenarios

Single Chip

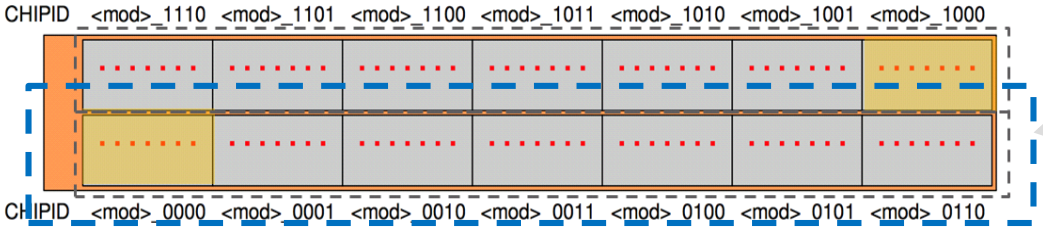
Readout,
Misc features

INNER BARREL *MODULE*



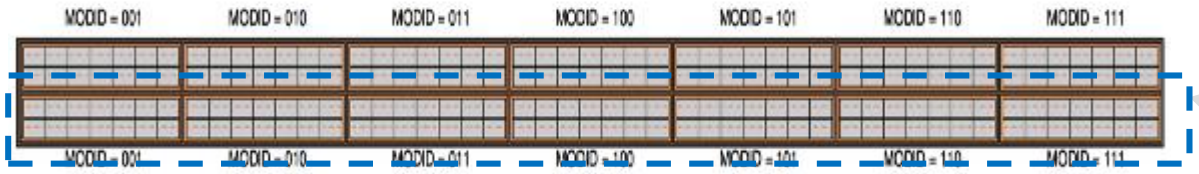
Control Features

OUTER BARREL *MODULE*



Inter-chip data
exchange protocol

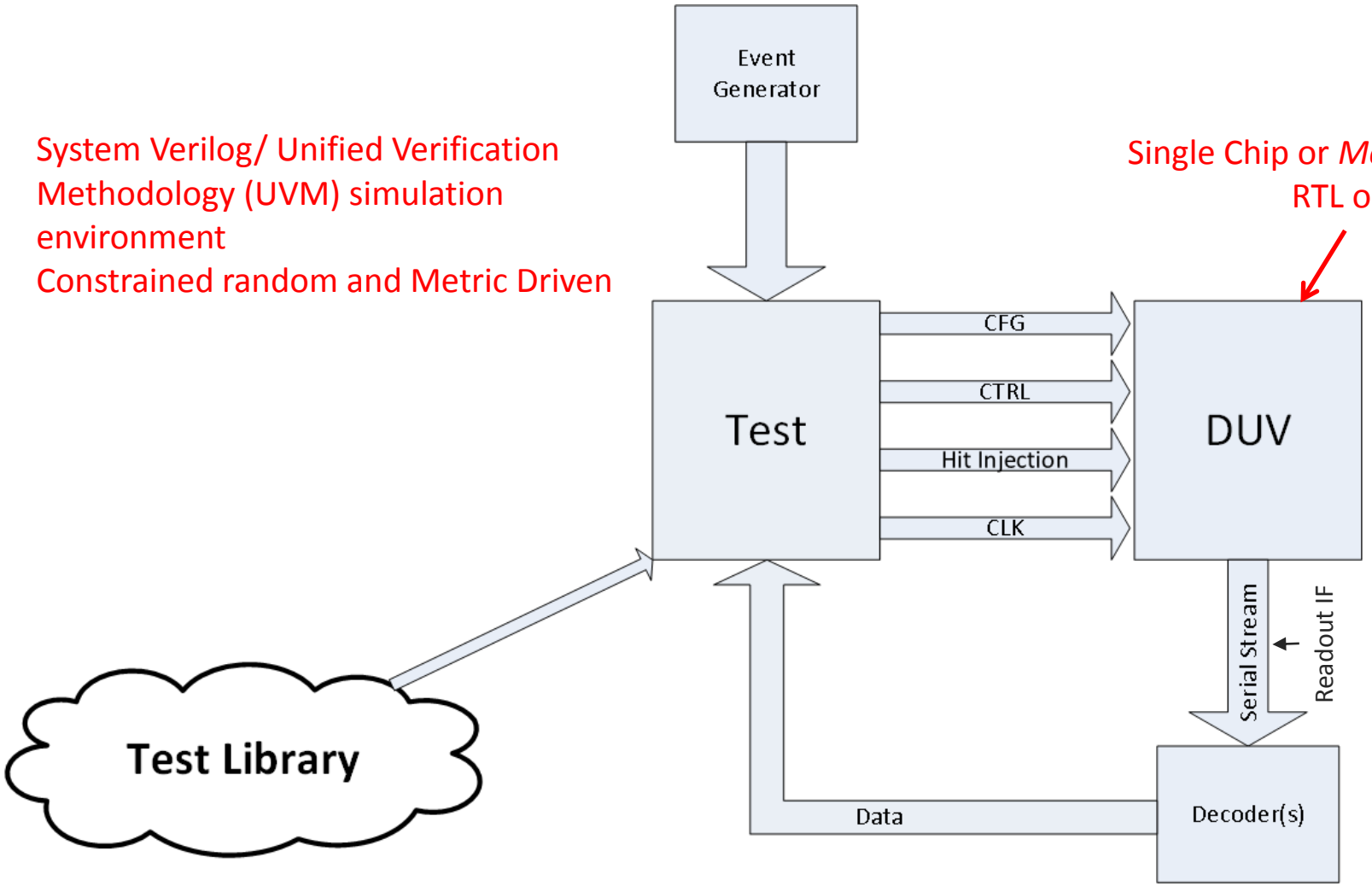
OUTER LAYER *STAVE*



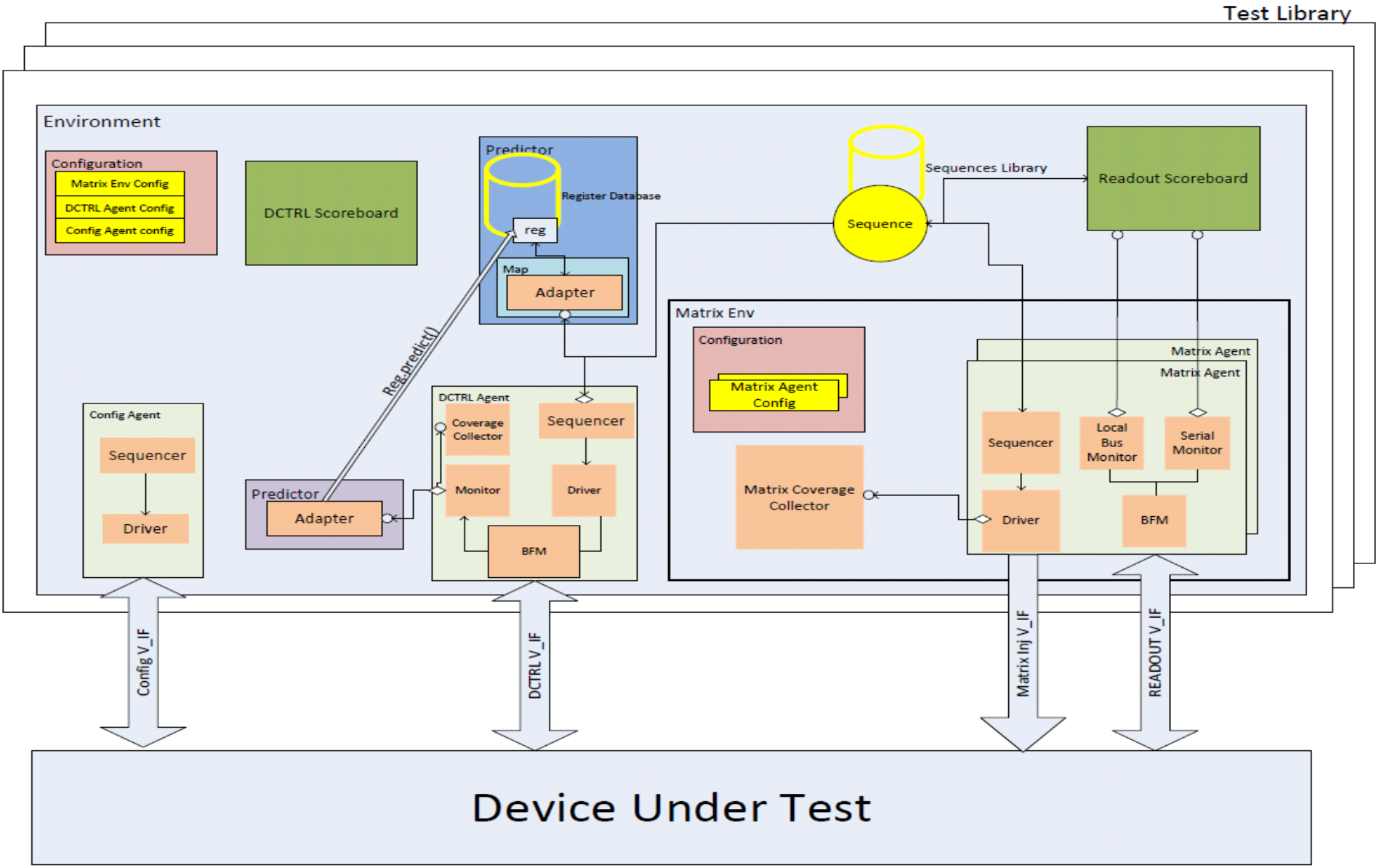
Control features, 49
chips!

TB Top view

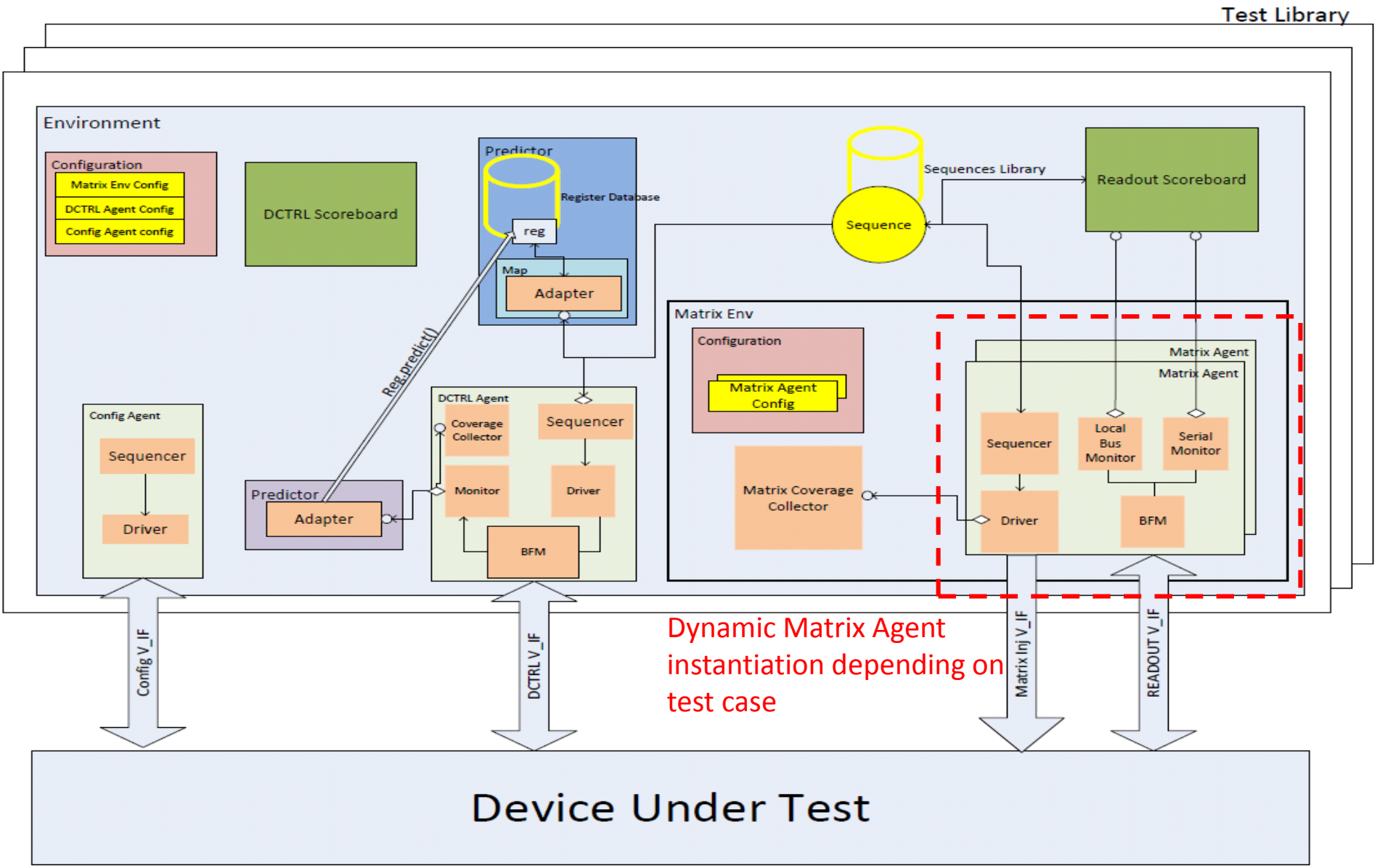
- System Verilog/ Unified Verification Methodology (UVM) simulation environment
- Constrained random and Metric Driven



TB Architecture - UVM



TB Architecture - UVM



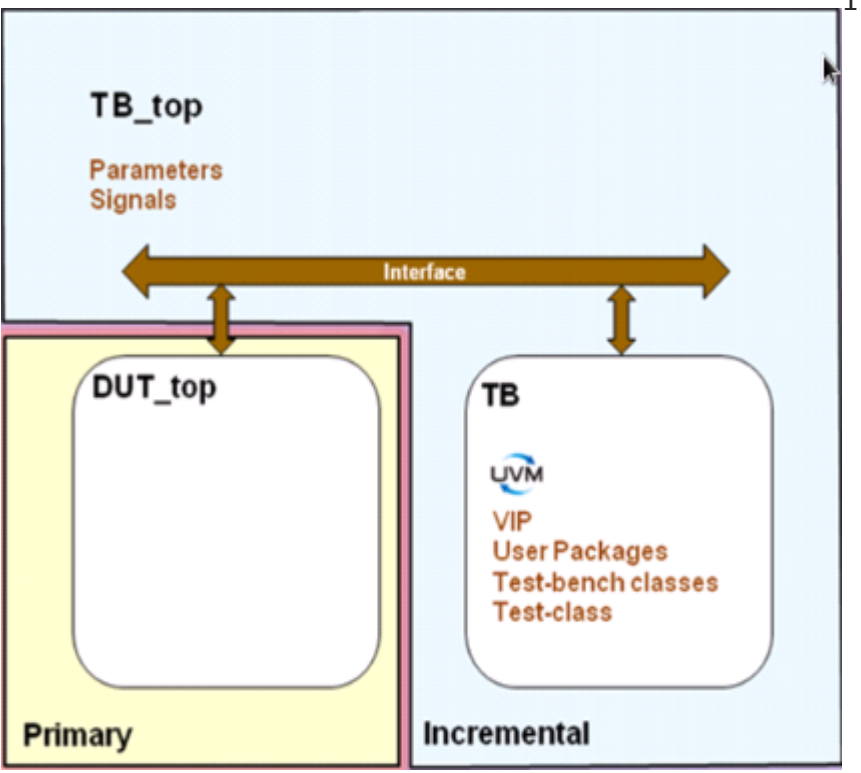
MULTI-SNAPSHOT INCREMENTAL ELABORATION

Problems to be Addressed

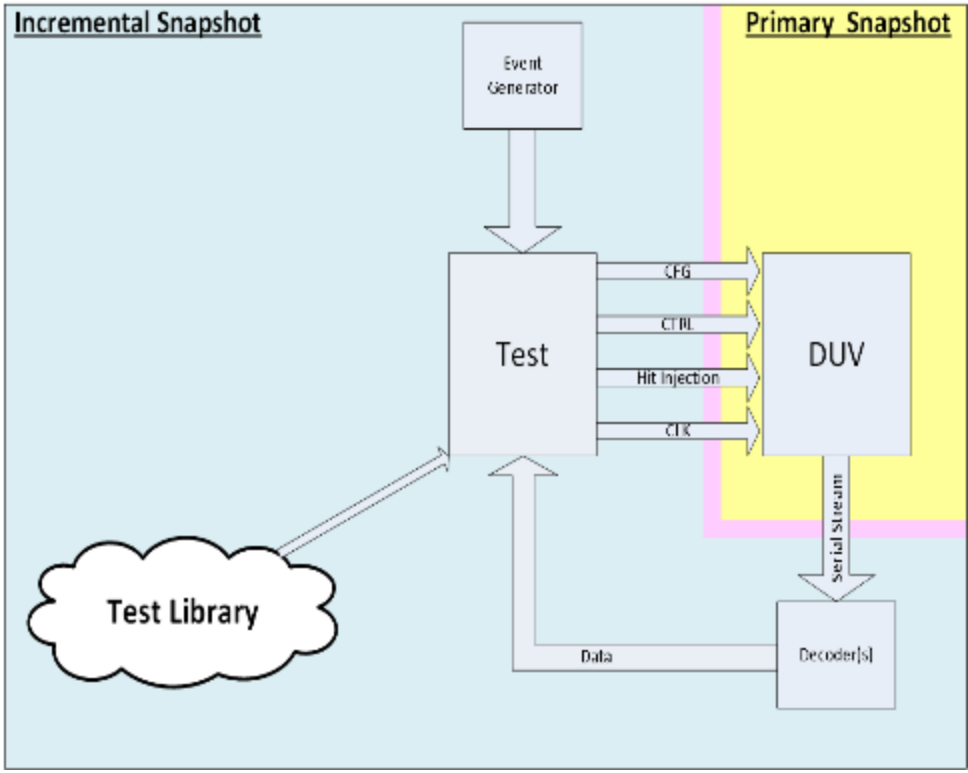
1. Multiple DUV scenarios to simulate
   - Dynamic, flexible software TB, could support any number of chips
   - Would need a new elaboration for every scenario
2. Every variation in test case means re-elaboration of complete snapshot
   - For ALPIDE, at least 7 minutes (simplest scenario)
   - Regression suites also suffer from additive delays due to this effect

Incremental Elaboration

- Elaborate Design Once
- Modify TB and VIP quickly



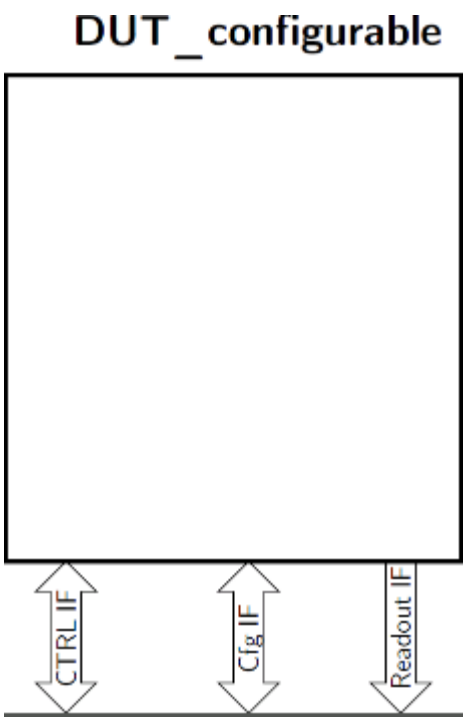
[1]



[1] Multi Snapshot Incremental Elaboration, Cadence, Product Version 14.1, June 2014

Top level TB

```
test_top test(.ctrl_if(ctrl_if),
              .cfg_if(cfg_if),
              .readout_if(readout_if),
              .trigger_filter(trigger_filter)
              );

DUT_configurable  CHIP_INST( .i_f(ctrl_if),
                             .cfg_if(cfg_if),
                             .readout_if(readout_if)
                             );
```

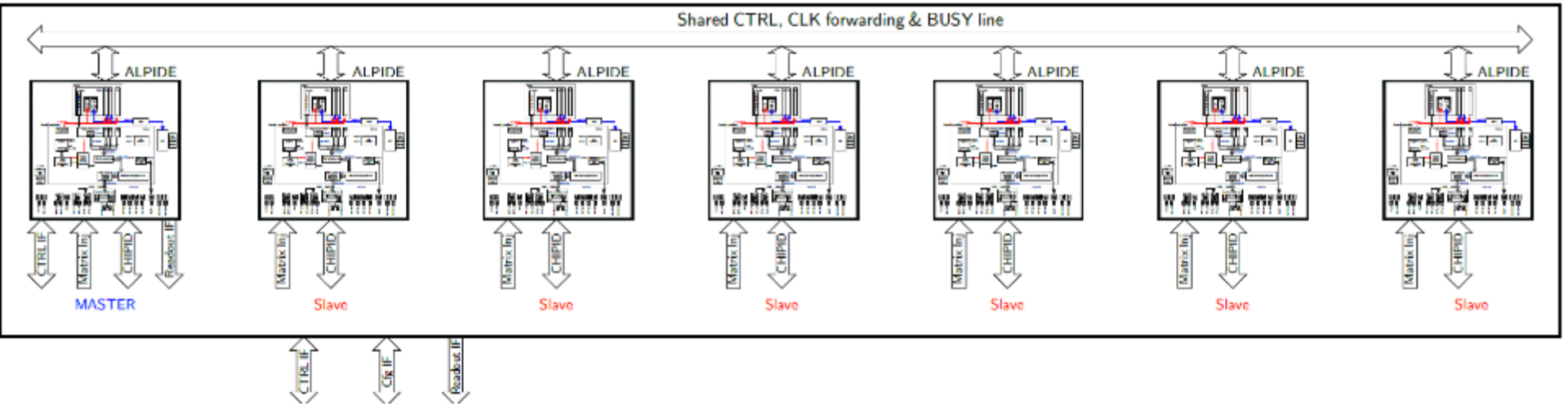


Placeholder Module, always replaced by DUV through libmap configuration

```
config cfg_dut_ob_7;
design TBs.top_tb;
default liblist matrix_model_structural TBs DUTs tsl18fs120;
cell DUT_configurable use DUTs.DUT_OB_full;
cell chip_unit_device_configurable use DUTs.chip_unit_device;
endconfig


config cfg_dut_ob_7_no_matrix;
design TBs.top_tb;
default liblist matrix_model_structural TBs DUTs tsl18fs120;
cell DUT_configurable use DUTs.DUT_OB_full;
cell chip_unit_device_configurable use DUTs.chip_unit_device_no_matrix;
endconfig


config cfg_dut_ib_single_chip;
design TBs.top_tb;
default liblist matrix_model_structural TBs DUTs tsl18fs120;
cell DUT_configurable use DUTs.DUT_IB_Single_Chip;
cell chip_unit_device_configurable use DUTs.chip_unit_device;
endconfig
```

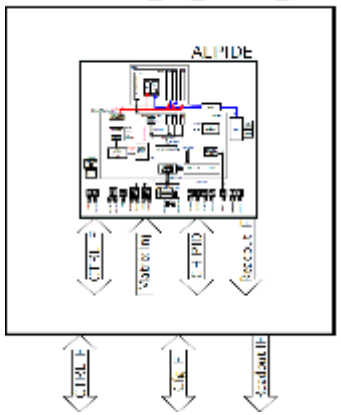
DUV – Configurations

DUT_OB_full

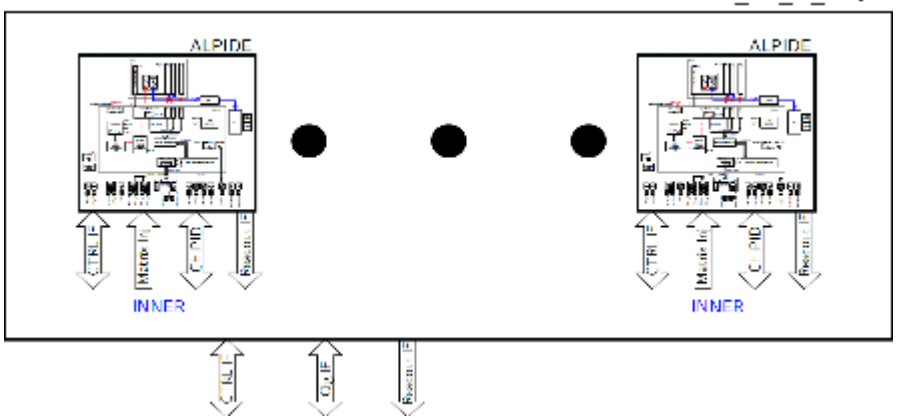


- Configuration specifics hidden on lower level
- DUV boundary interface remains the same

DUT_IB_Single_Chip

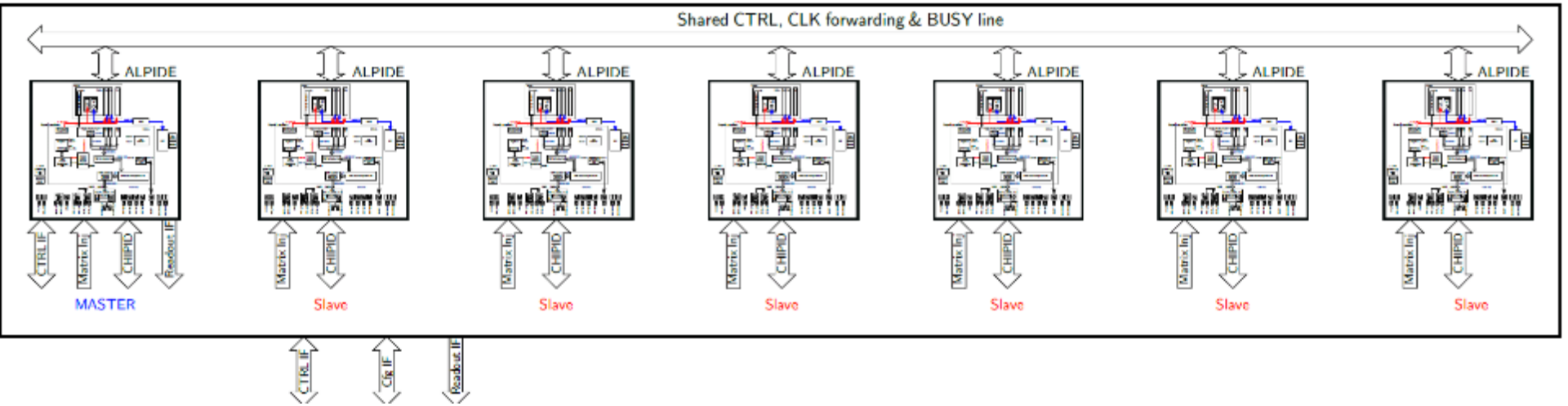


DUT_IB_9_chips



Example of Elaboration Gains

DUT_OB_full

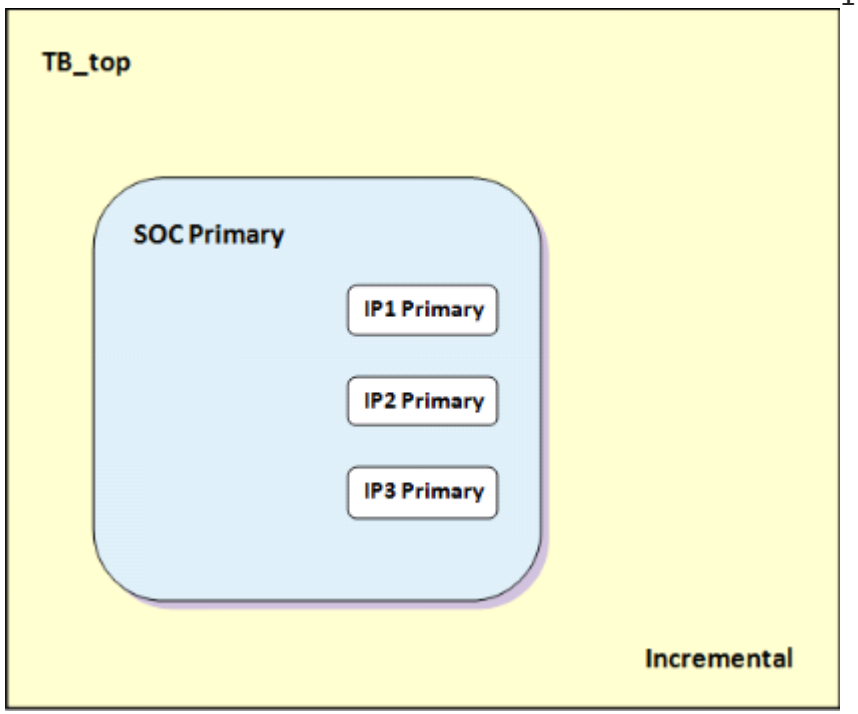


Metrics	Multi-Snapshot Incremental Elaboration	Typical DUT + TB elaboration
Elaboration Time	Total : 8m55s (6m58s prim elab + 1m57s cloning and loading TB)	43m4s
Final Snapshot loaded in Memory	33.5 GB	24 GB

BIGGEST ADVANTAGE: TB DEVELOPMENT SPEED-UP

Note: Configuration above uses Periphery RTL, Matrix & I/O behavioral models

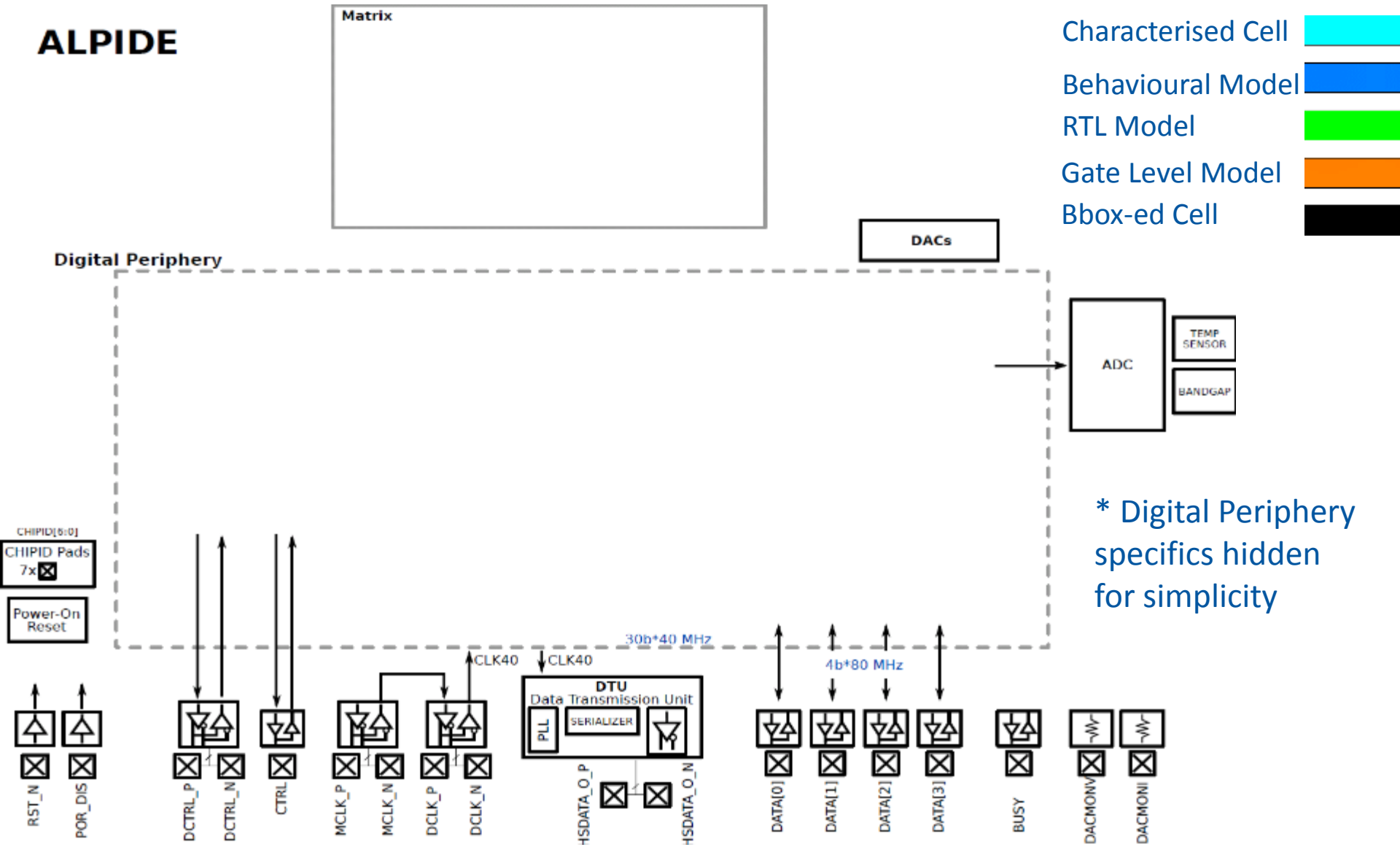
Can we improve further?



- Modular design
- Suggested approach of building up SoC through separately elaborated IP

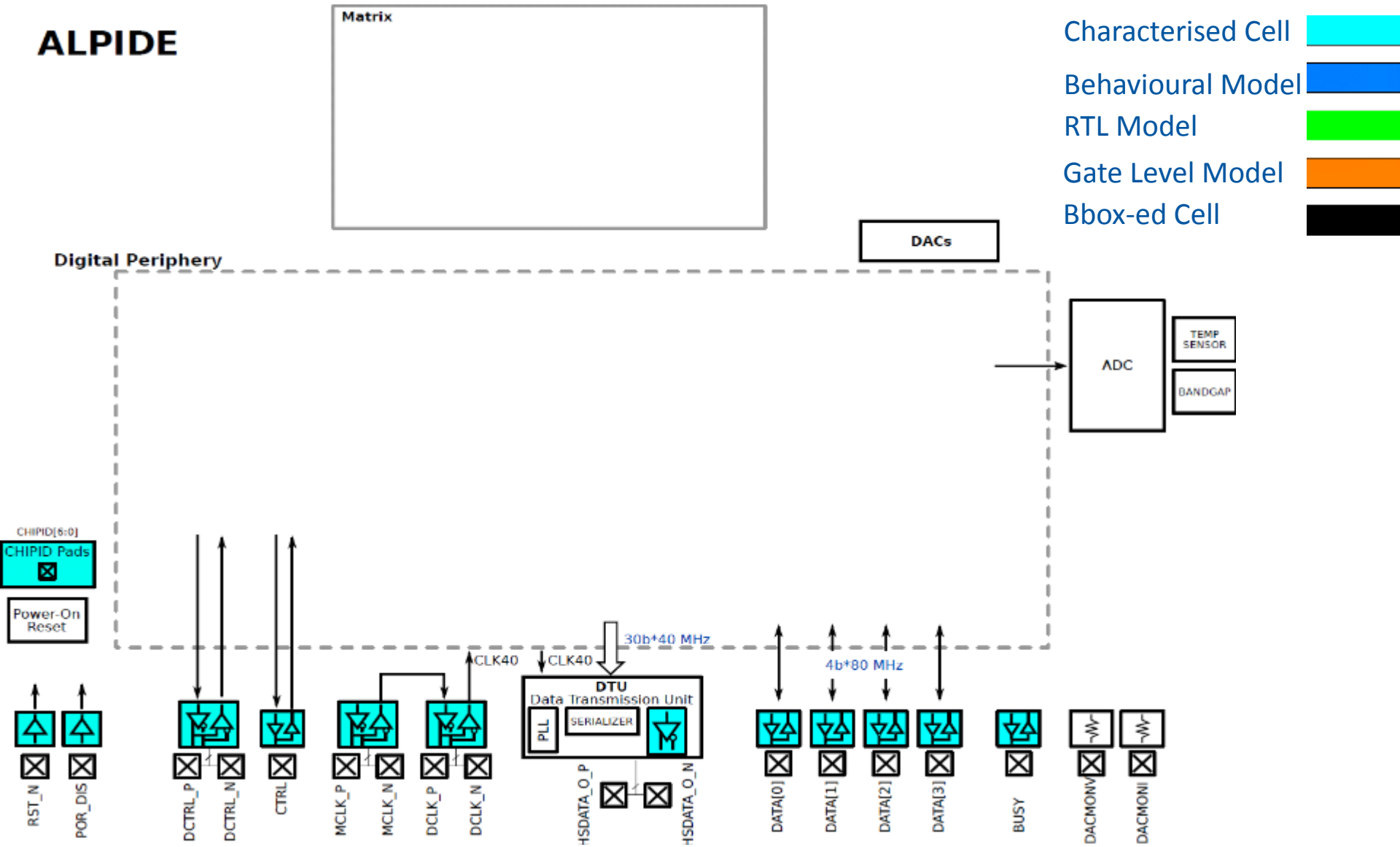
[1] Multi Snapshot Incremental Elaboration, Cadence, Product Version 14.1, June 2014

Top Level Blocks within the ALPIDE

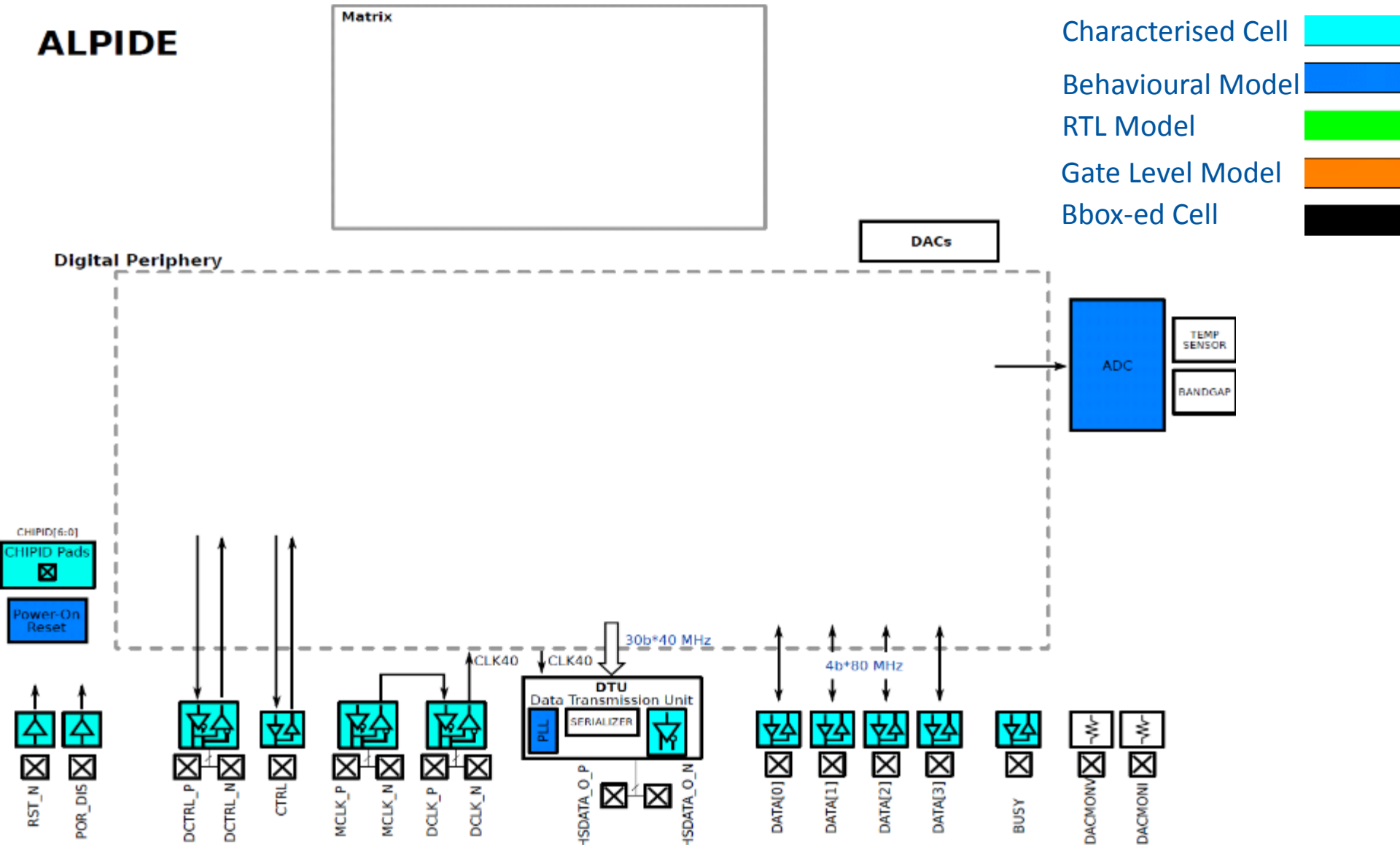


* Digital Periphery specifics hidden for simplicity

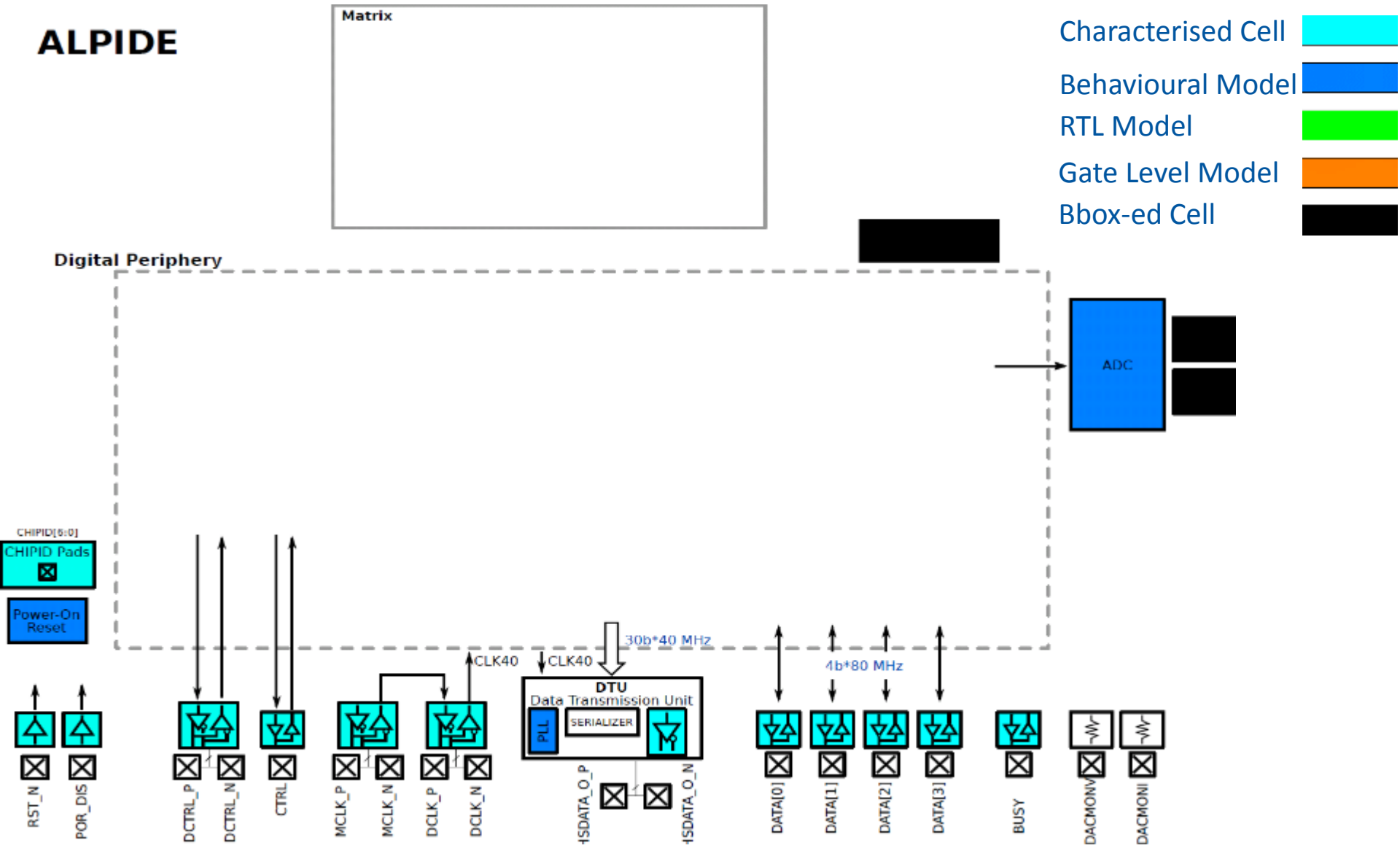
I/Os – Characterised Cells



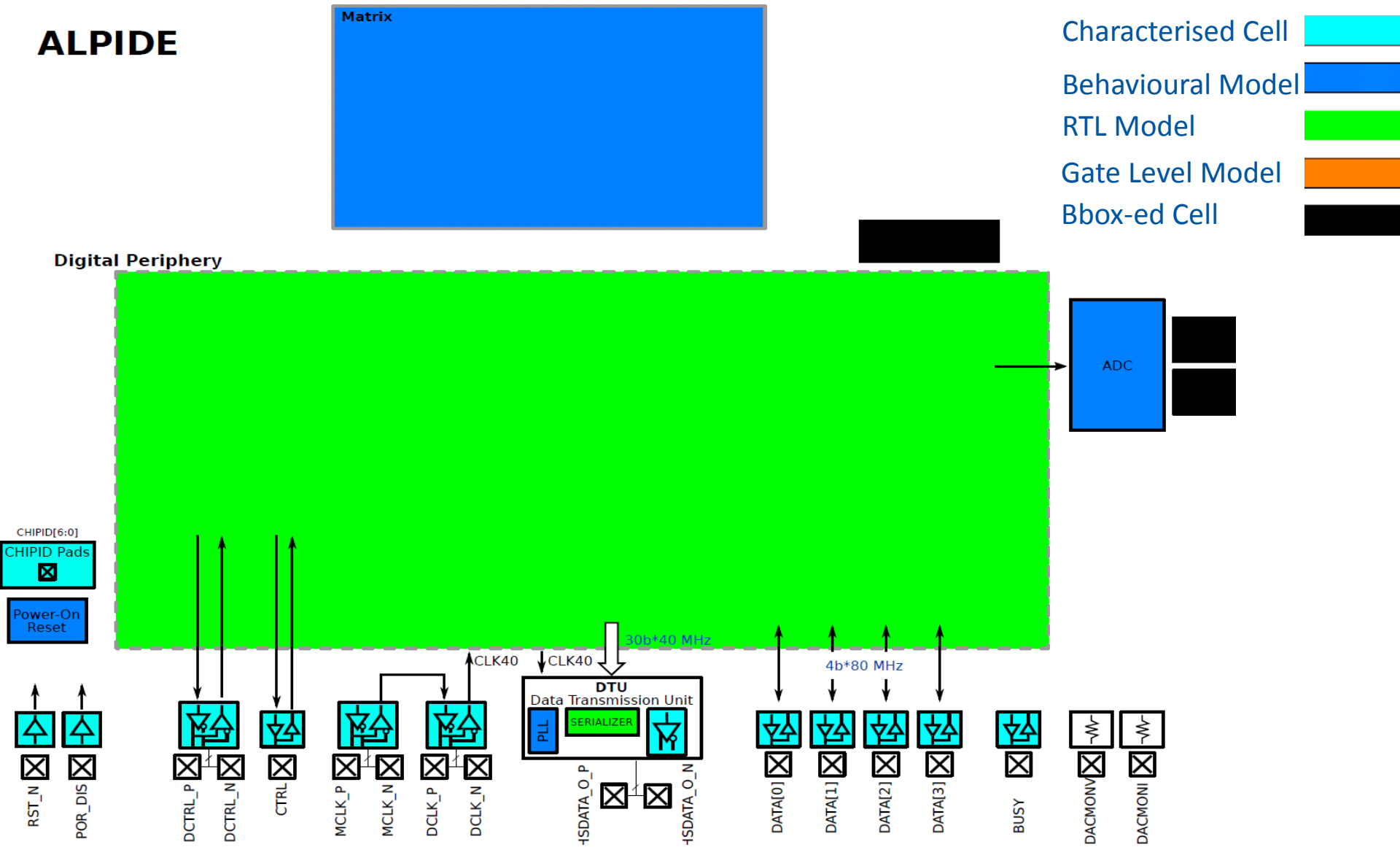
Behavioural Models Added



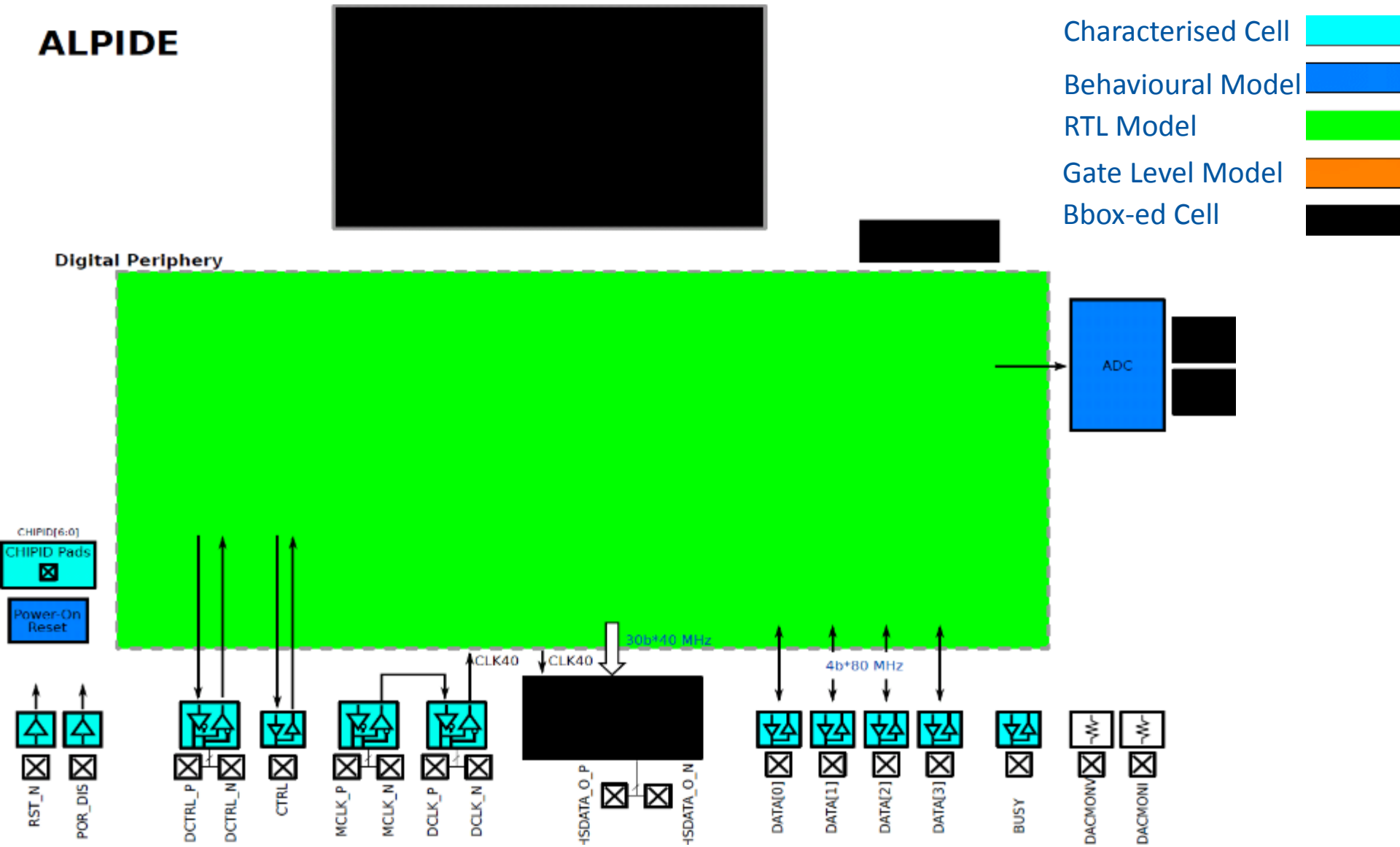
Analog Blocks Bbox-ed



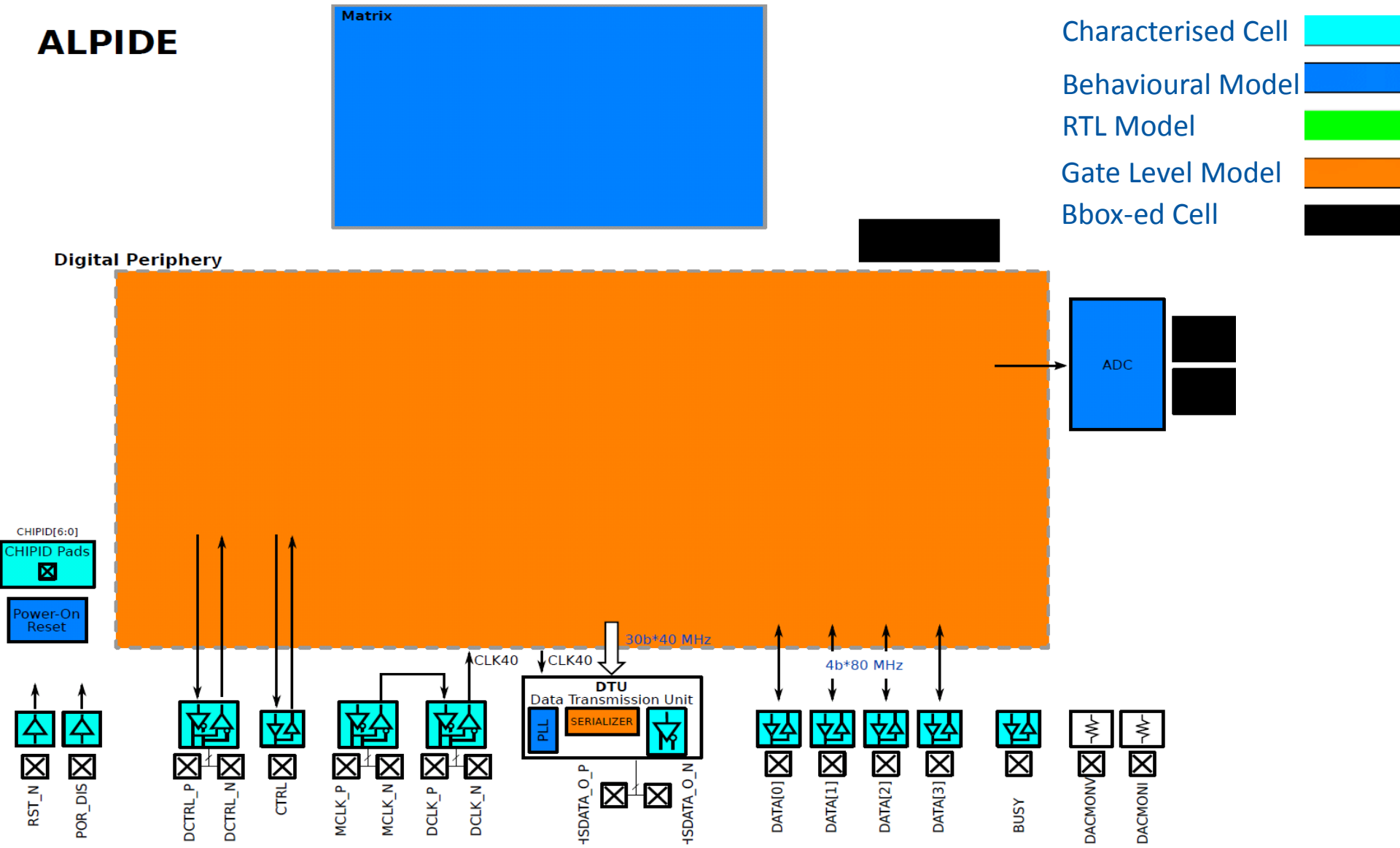
RTL - Option 1


ALPIDE
Matrix
Characterised Cell
Behavioural Model
RTL Model
Gate Level Model
Bbox-ed Cell
Digital Periphery
ADC
CHIPID[6:0]
CHIPID Pads
Power-On Reset
30b*40 MHz
CLK40
CLK40
4b*80 MHz
DTU
Data Transmission Unit
PLL
SERIALIZER
RST_N
POR_DIS
DCTRL_P
DCTRL_N
CTRL
MCLK_P
MCLK_N
DCLK_P
DCLK_N
LDATA_O_P
LDATA_O_N
DATA[0]
DATA[1]
DATA[2]
DATA[3]
BUSY
DACMONV
DACMONI

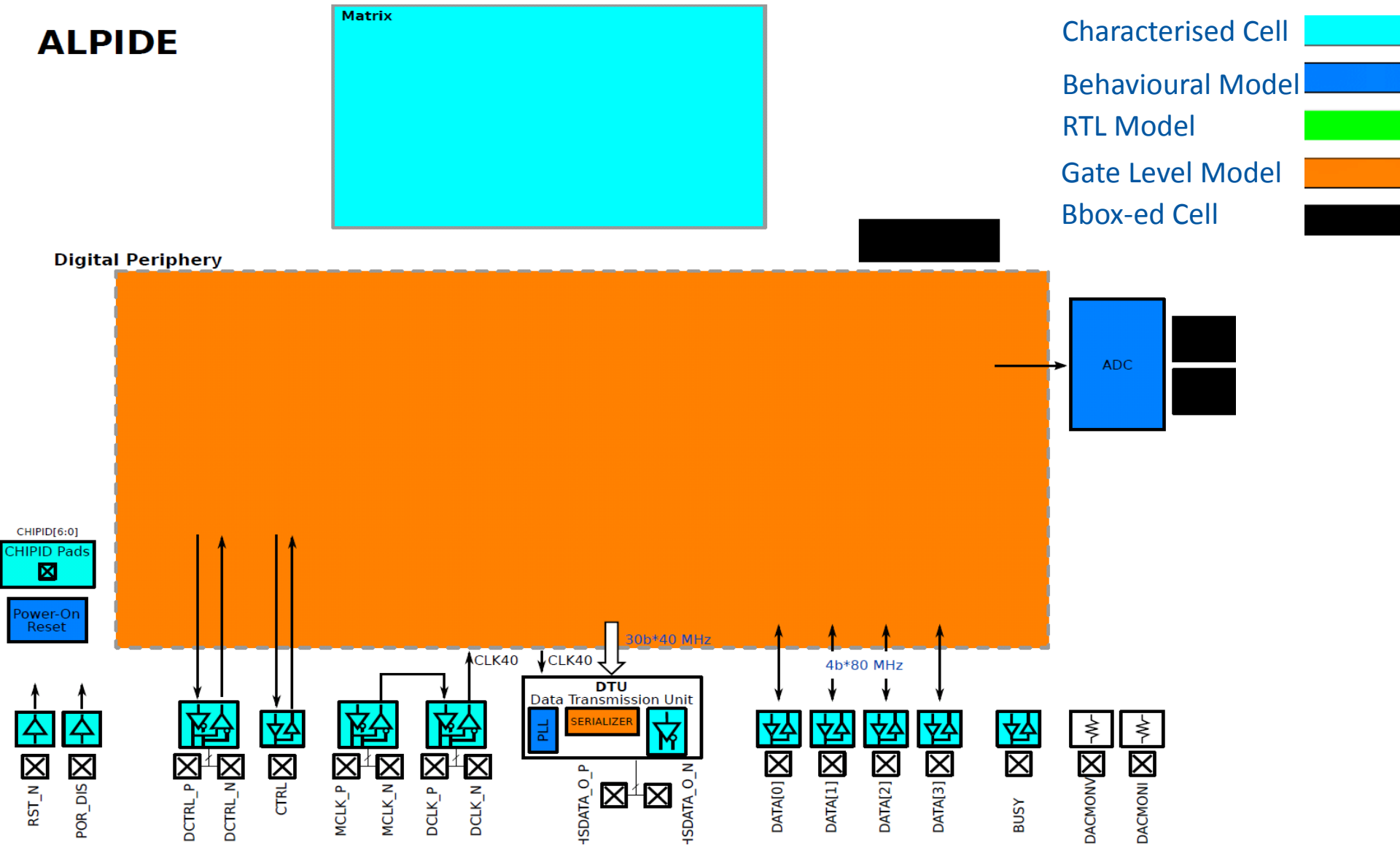
RTL - Option 2


ALPIDE
Characterised Cell
Behavioural Model
RTL Model
Gate Level Model
Bbox-ed Cell
Digital Periphery
ADC
CHIPID[6:0]
CHIPID Pads
Power-On Reset
CLK40
CLK40
30b*40 MHz
4b*80 MHz
RST_N
POR_DIS
DCTRL_P
DCTRL_N
CTRL
MCLK_P
MCLK_N
DCLK_P
DCLK_N
HSDATA_O_P
HSDATA_O_N
DATA[0]
DATA[1]
DATA[2]
DATA[3]
BUSY
DACMONV
DACMONI

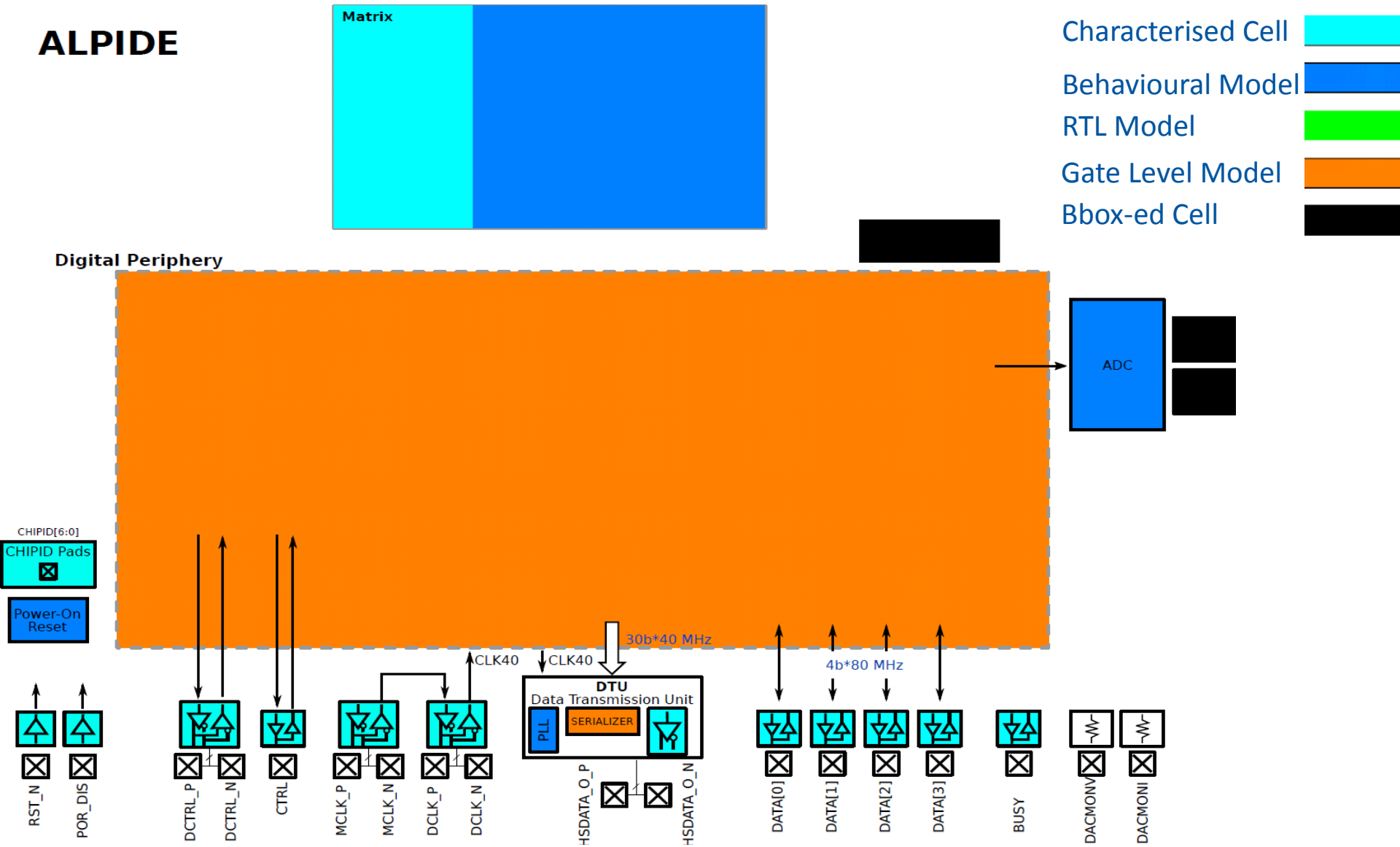
Gate Level - Option 1



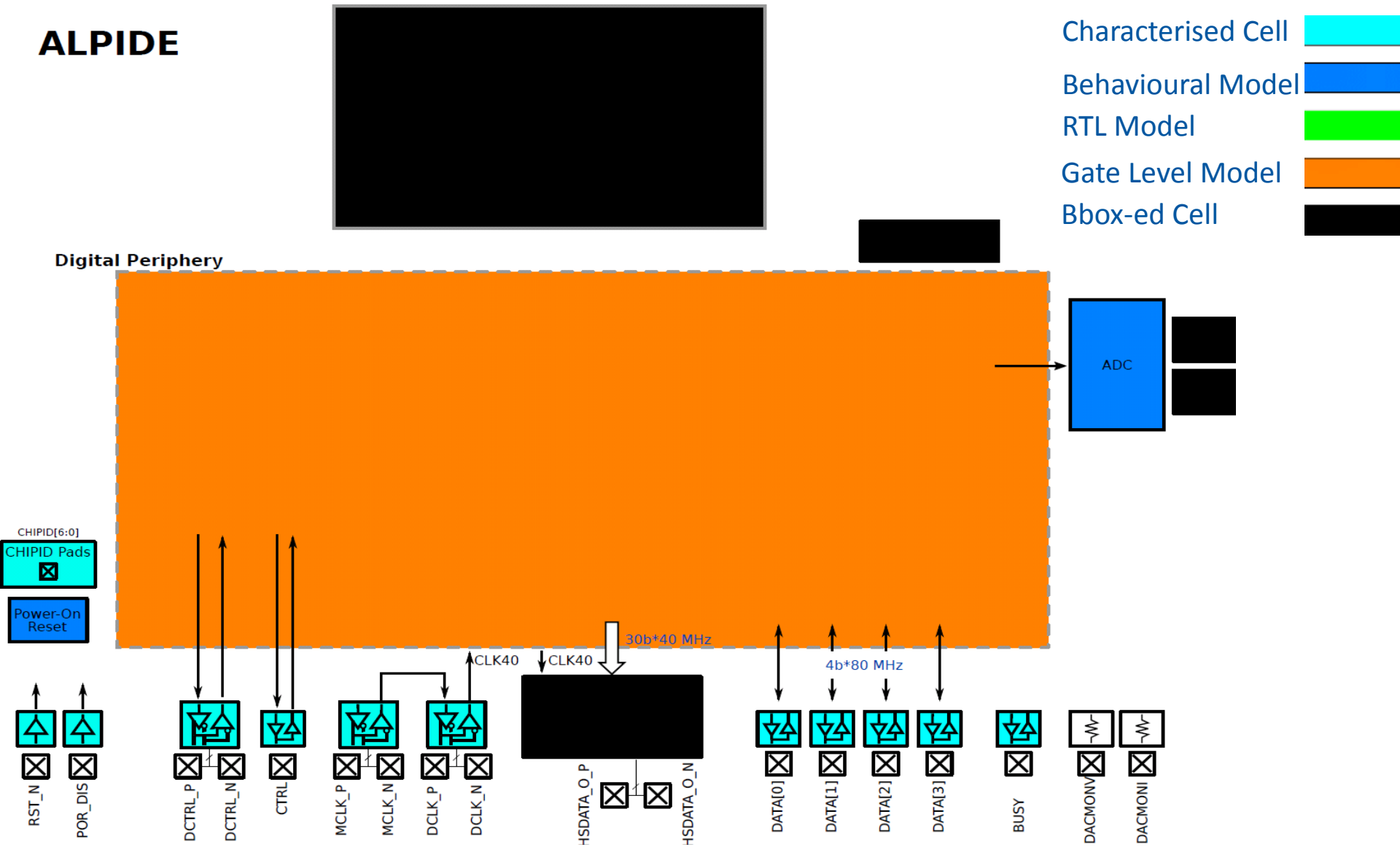
Gate Level – Option 2



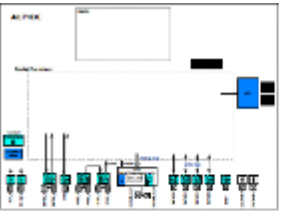
Gate Level – Option 3



Gate Level – Option 4



Individual Chip Config



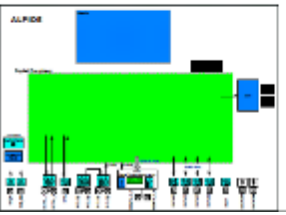
Characterised Cell

Behavioural Model

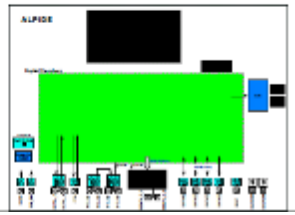
RTL Model

Gate Level Model

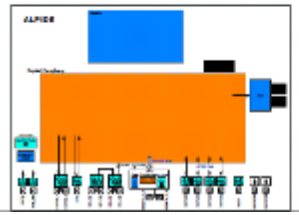
Bbox-ed Cell



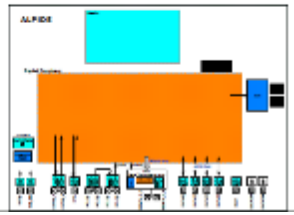
RTL – Option 1
Elaboration time A: 8m55s
Elaboration time B: 43m4s



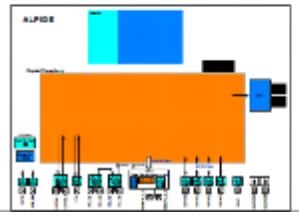
RTL – Option 2
Elaboration time A: 33s
Elaboration time B: 5m16s



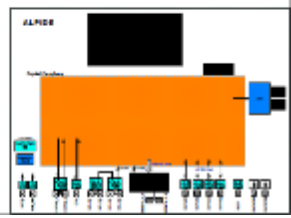
Gate – Option 1
Elaboration time A: 10m52s
Elaboration time B: 55m4s



Gate – Option 2
Elaboration time A: 36m34s
Elaboration time B: **>1h**



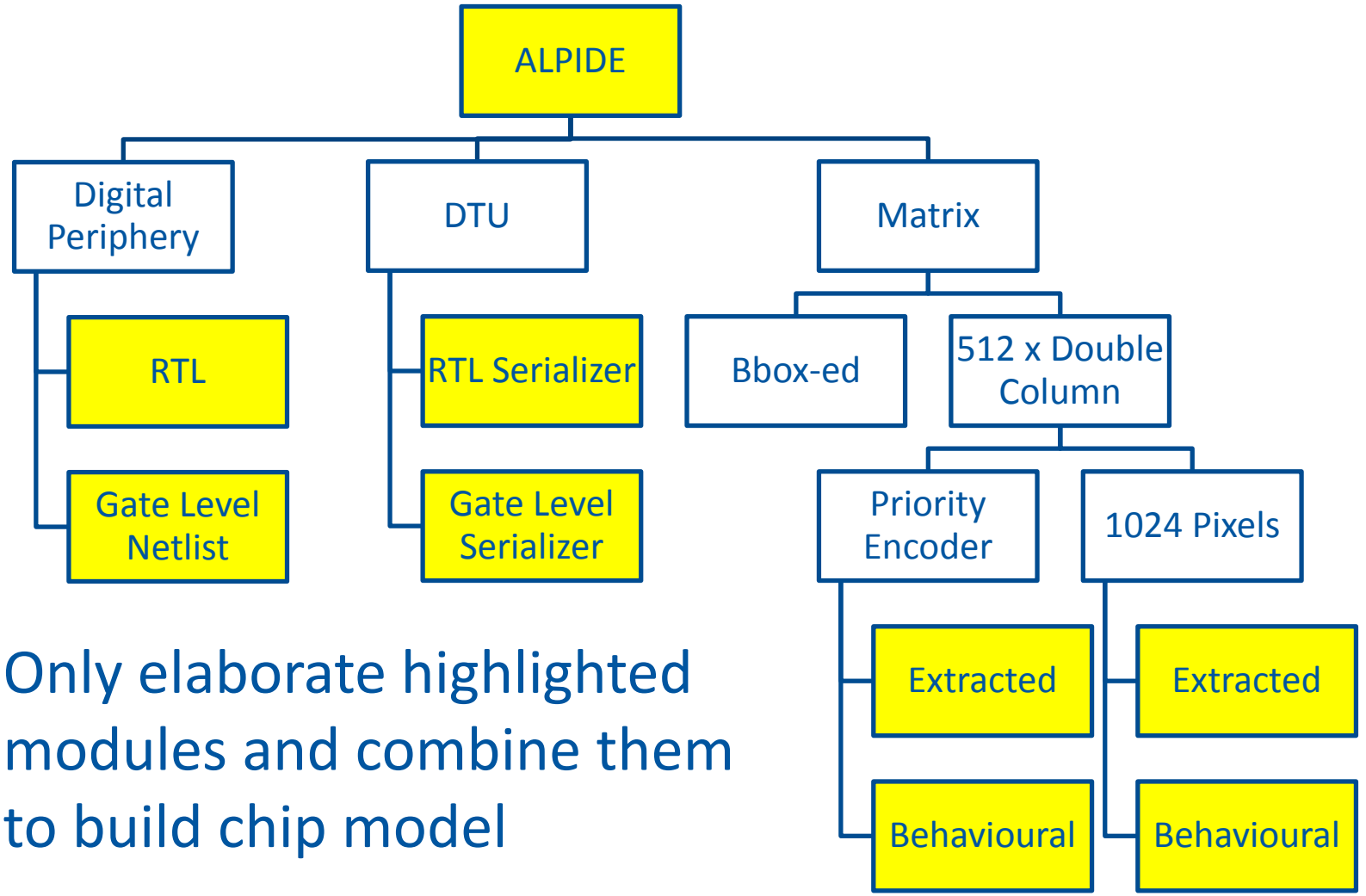
Gate – Option 3
Elaboration time A: 14m21s
Elaboration time B: **>1h**



Gate – Option 4
Elaboration time A: 1m49s
Elaboration time B: 8m33s

Note: Elaboration times given for the reference scenario of OB (7 chips shown in slide 24).
Elaboration A corresponds to Multi Snapshot Incremental Elaboration
Elaboration B corresponds to typical all-in-one Elaboration

Summary – Primary Snapshots



Only elaborate highlighted modules and combine them to build chip model

Summary of Advantages

- Running regressions
- Pick and choose compromise between performance and accuracy
- Easier to bind checkers to primary snapshot

ASSERTION BASED VERIFICATION (ABV) AND FORMAL

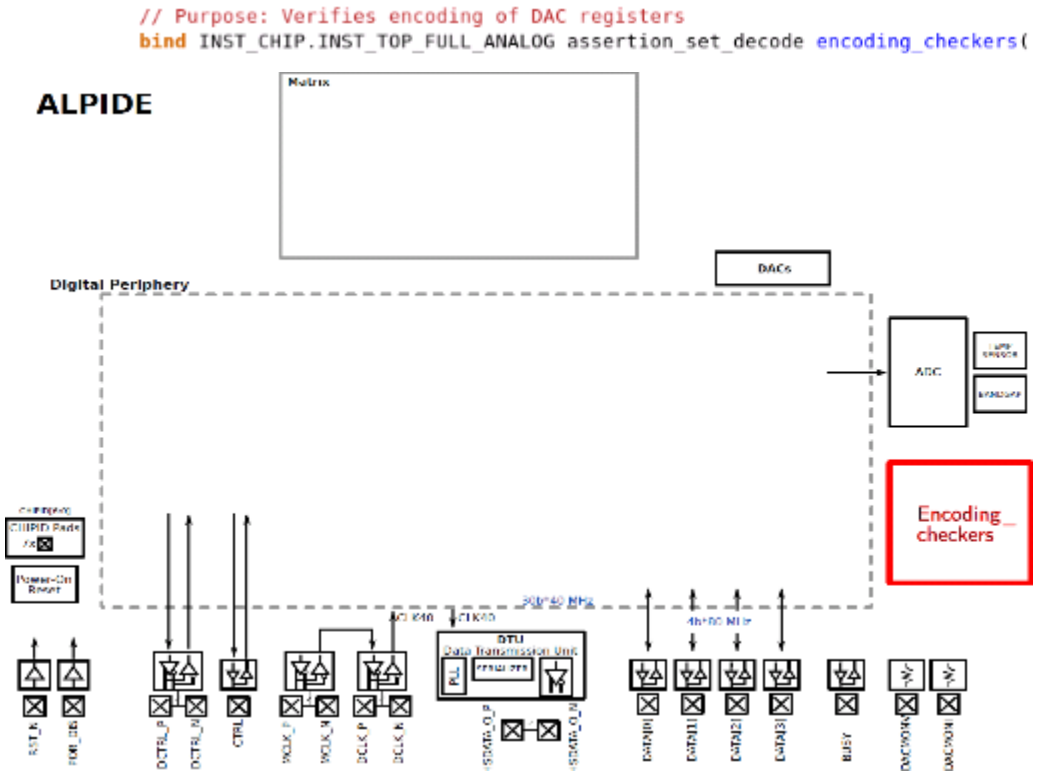
Objective

Show through examples the value of ABV :

- Formalise specifications on block level
- Facilitate regression checks
- Accelerate development

Binding Instances Intro

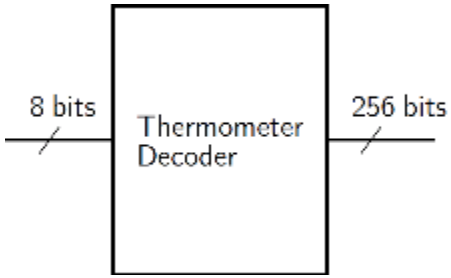
- Interfaces / Modules bound to DUT's hierarchy
- Contain checks on interfaces, protocol as well as coverage metrics

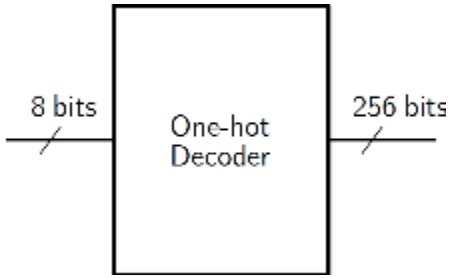



```
// Purpose: Verifies encoding of DAC registers
bind INST_CHIP.INST_TOP_FULL_ANALOG assertion_set_decode encoding_checkers( .SET_VCASN(SET_VCASN),
                                                                             .SET_VCASN2(SET_VCASN2),
                                                                             .SET_VCASP(SET_VCASP),
                                                                             .SET_VCLIP(SET_VCLIP),
                                                                             .SET_VPLSE_HIGH(SET_VPLSE_HIGH),
                                                                             .SET_VPLSE_LOW(SET_VPLSE_LOW),
                                                                             .SET_VRESET_P(SET_VRESET_P),
                                                                             .SET_VRESET_D(SET_VRESET_D),
                                                                             .SET_VTEMP(SET_VTEMP),
                                                                             .SET_IRESET(SET_IRESET),
                                                                             .SET_IBIAS(SET_IBIAS),
                                                                             .SET_ITHR(SET_ITHR),
                                                                             .SET_IDB(SET_IDB),
                                                                             .SET_IAUX2(SET_IAUX2),
                                                                             .SWCNTL_IREF_BUFFER(SelIdc[2]),
                                                                             .SWCNTL_ADCDAC(INST_CHIP.INST_PAD_RING.ADC_SEL_DAC_OUT),
                                                                             .*);
```

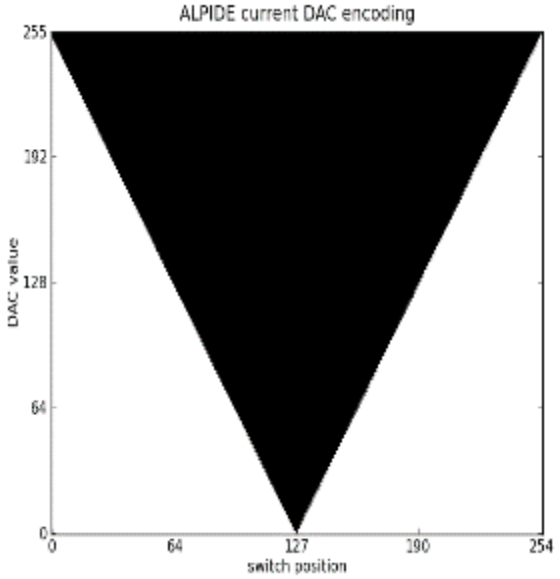
Example 1 : DAC Decoders

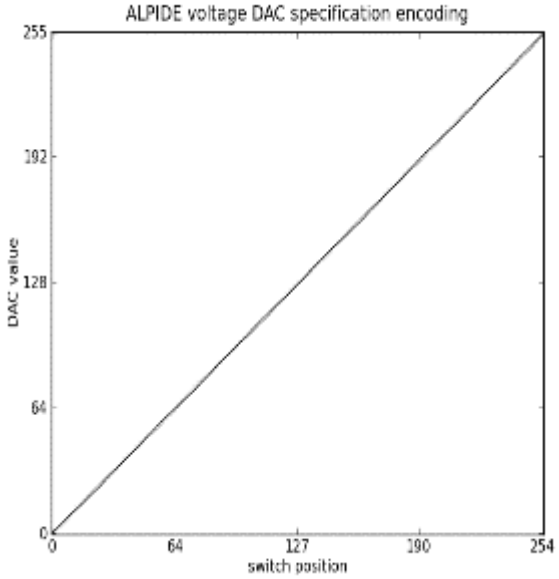
Specification





How do we ensure specification and regression protection?





Note: Current switches enable identical sources. Switches spread along the length of the chip

Checks Encoding on Change

General Properties

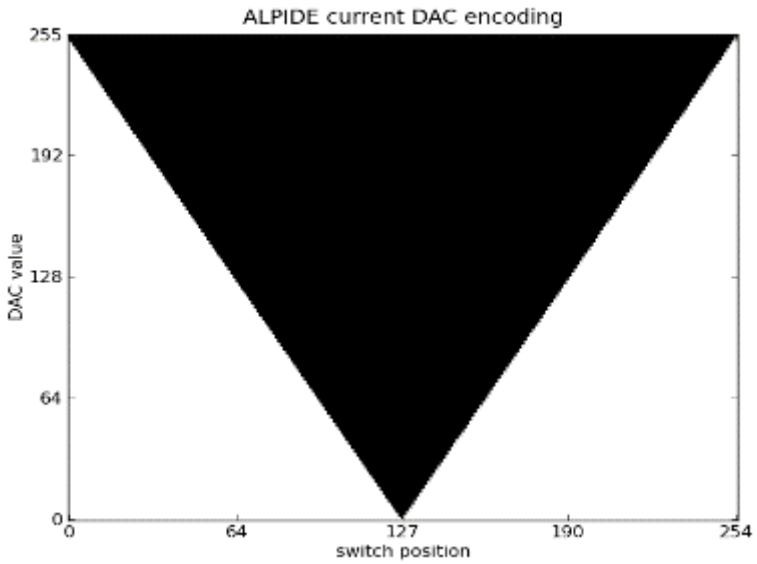
```
// For IDACs, thermometer code
property Therm_encoding_check(bit [7:0] DAC_value, bit [255:0] encoded_value);
   @cb (DAC_value == $countones(encoded_value)) && (encoded_value == decode_expected_thermometer_code(DAC_value));
endproperty

// For VDACs, $onehot0 returns 1 if at most one bit is high, clog2 checks that we have hte correct bit being high
property  Onehot_encoding_check(bit [7:0] DAC_value, bit [255:0] encoded_value);
   @cb (DAC_value == $clog2(encoded_value)) && $onehot0(encoded_value);
endproperty
```

Instantiation

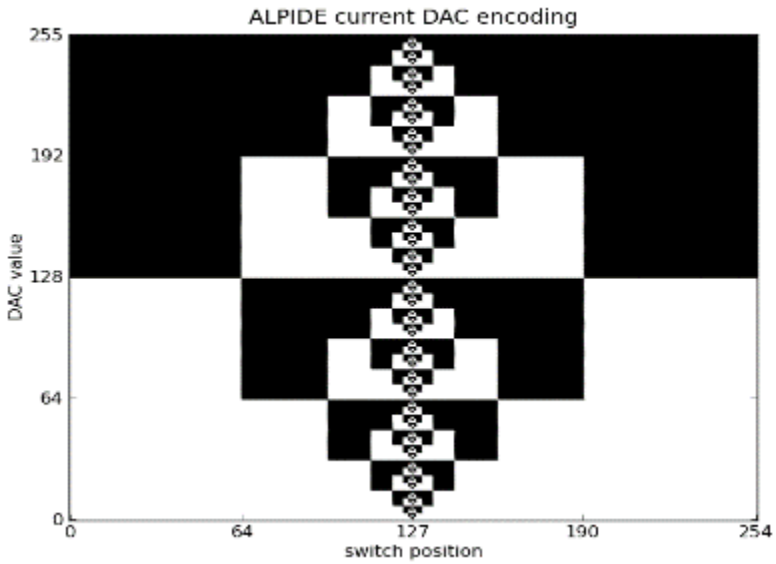
```
// // IDACS
CHECK_ITHR: assert property(Therm_encoding_check(.DAC_value(ITHR),.encoded_value(SET_ITHR)))
  else $error("Analogue Decoding Failure Analogue Encoding Failure - ITHR");


// VDACS
CHECK_VCASP: assert property(Onehot_encoding_check(.DAC_value(VCASP),.encoded_value(SET_VCASP)))
  else $error("Analogue Decoding Failure Analogue Encoding Failure - VCASP");
```

Encoding Suboptimal Implementation Spotted

Desired Thermometer



Implemented “Thermometer”



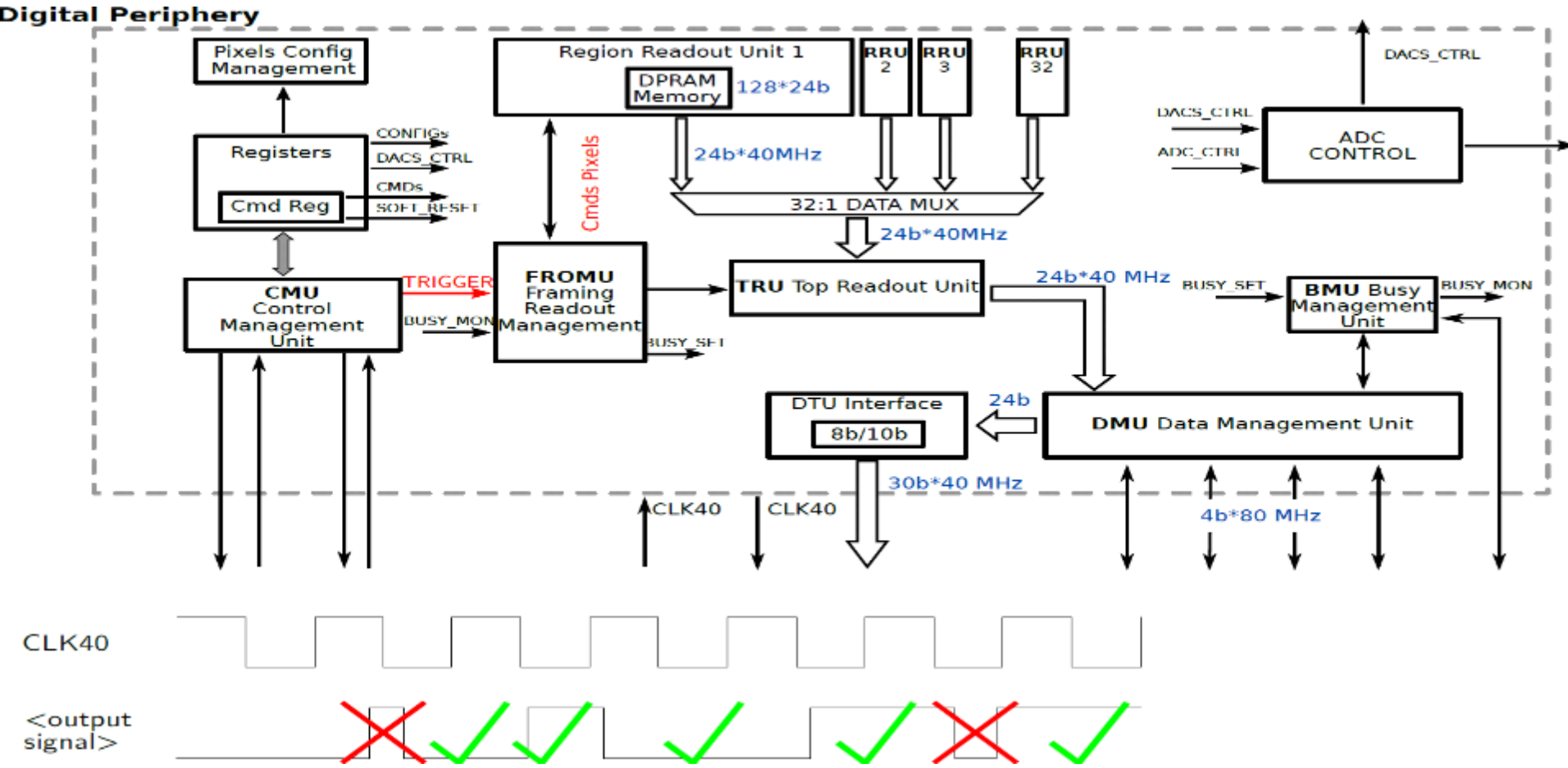
Identical Number of Switches on for every DAC value!

- Checks easy to implement and integrate, yet powerful
- Runs in background during Simulation, always => good for regression

Example 2 : Detecting glitches

Specification

There shall be no pulses shorter than half a clock period (glitches) at the boundary of the digital periphery.



Detecting glitches

Detecting glitches at Digital Periphery borders – simple but powerful application (NOTE: Gate Level only, always time-annotated)

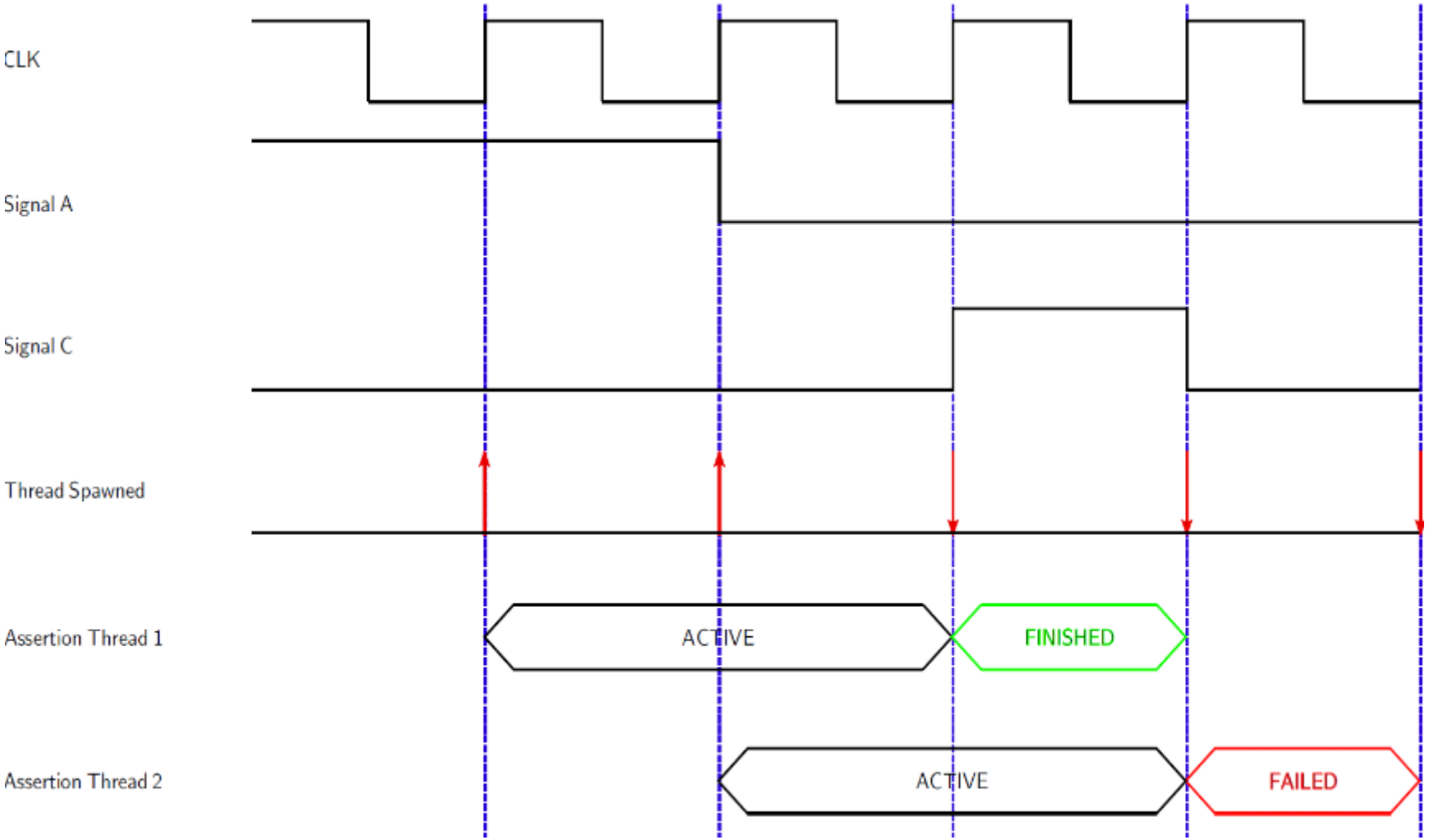
```
property glitch_1bit_half_period(bit input_signal);
   real first_change;
   @(input_signal)  disable iff (!chip_rst_n) ( 1 , first_change=$realtime) |=> ( ($realtime - first_change) > percantage_glitch_cutoff * clk_half_period );
endproperty
```

- Triggered on signal change
- Checks that the next change only occurs after a predefined duration
- Easy to do for all signals at boundary b/n digital & analogue

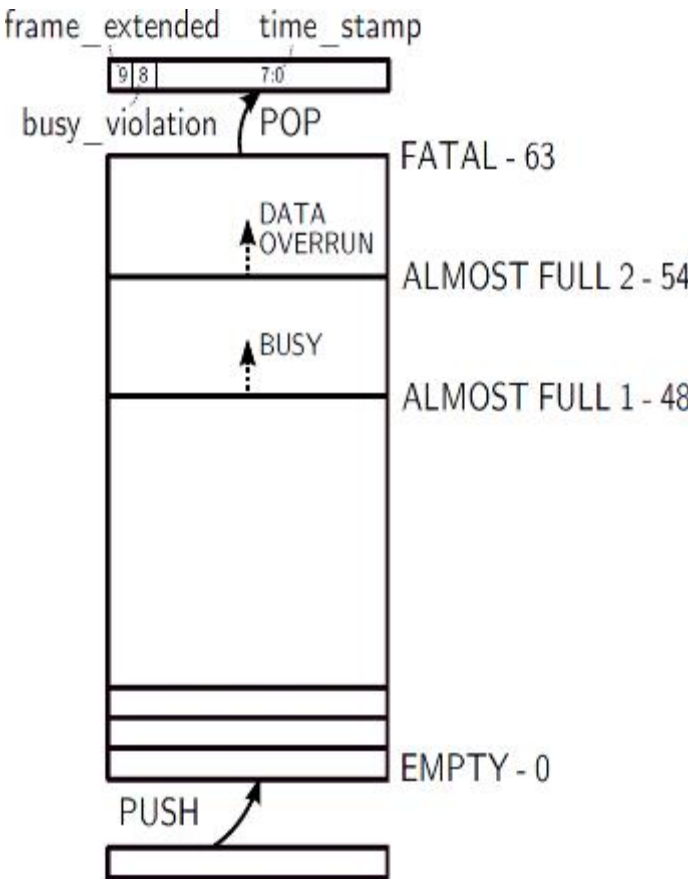
Spotted a GLITCH on a DRIVER EN signal!

Temporal Behaviour Example

A |-> !C [*2]



FIFO Example



Specification: There shall be no OVERFLOW or UNDERFLOW condition occurring

```
//  Instances

// Assertions
WRITE_WHEN_FULL: assert property(two_signals_illegal_delay(.first_signal(full),
                                                           .second_signal(write_en),
                                                           .delay(0),
                                                           .disable_signal(0)))
  else $error("%m Write following assertion of full");

RD_WHEN_EMPTY: assert property(two_signals_illegal_delay(.first_signal(empty),
                                                         .second_signal(read_en),
                                                         .delay(0),
                                                         .disable_signal(0)))
  else $error("%m RD when empty asserted");


   // Covers
CHECK_WR_AND_READ: cover property(two_signals_programmable_delay(.first_signal(write_en),
                                                                 .second_signal(read_en),
                                                                 .delay(0),
                                                                 .disable_signal(0)));
CHECK_FULL_ASSERTED: cover property(check_duration_single_bit_atleast(.line(full),
                                                                      .duration(1),
                                                                      .disable_signal(0)));
CHECK_EMPTY_ASSERTED: cover property(check_duration_single_bit_atleast(.line(empty),
                                                                       .duration(1),
                                                                       .disable_signal(0)));
```

Property Modularity – easy reusability

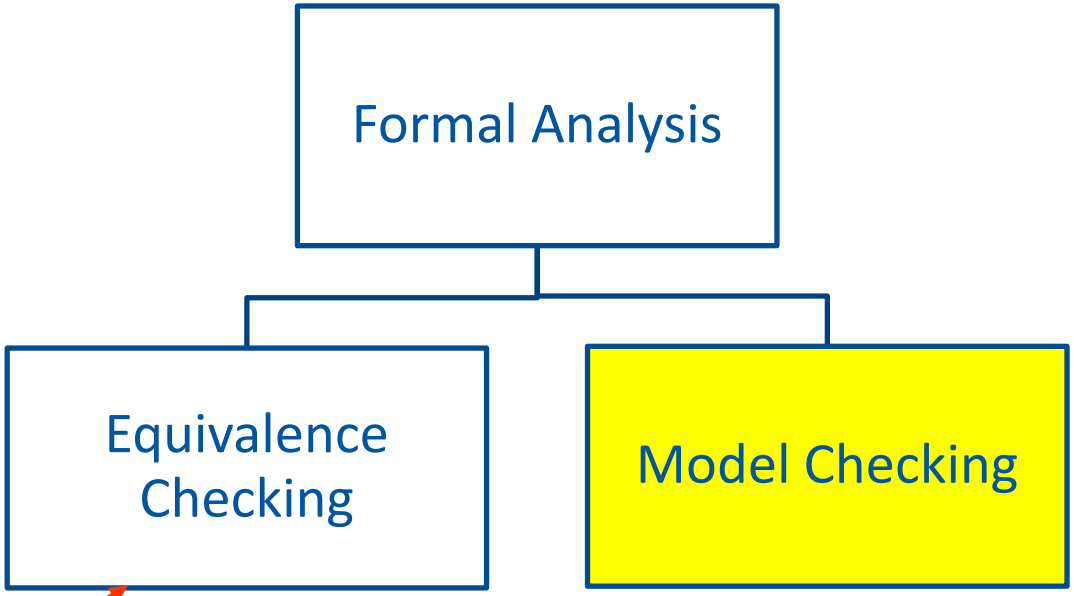
Generalise desired temporal behaviours and reuse

Property
two_signals_programmable_delay
two_signals_illegal_delay
first_signal_delayed_version_of_second_programmable
check_duration_single_bit_exact
check_duration_single_bit_atleast

Approach used by designers to capture and ensure engineering intent throughout the ALPIDE development

Formal Analysis Introduction

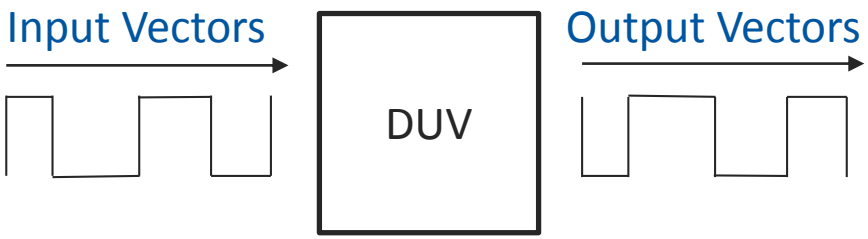
Formal Analysis is the act of comprehensively proving that a design meets the design intent



Ensures that two versions of design are functionally identical

Ensures that design matches specifications as defined by a set of properties

Simulation

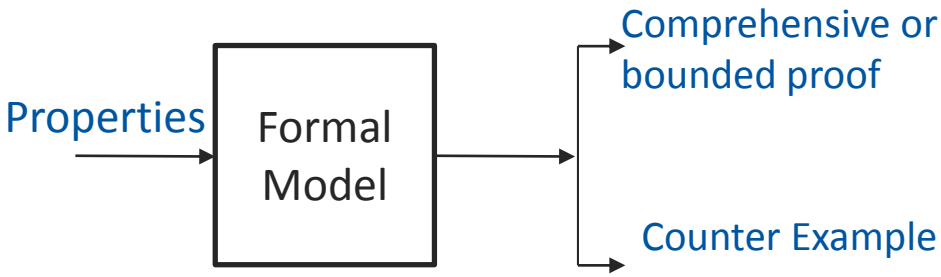


Pros:
- Able to verify large designs, SoC
- Can focus on scenarios of interest

Cons:
- Cannot explore the full state space
- Could miss corner cases
- Must design TB and stimulus
- Need to track root cause of bugs

Computational Challenges:
- Clock cycle limited

Formal



Pros:
- Able to explore the complete state space
- All corner cases caught
- No need to develop a TB, tool toggles inputs according to specified rules
- Tool pinpoints root cause of bugs

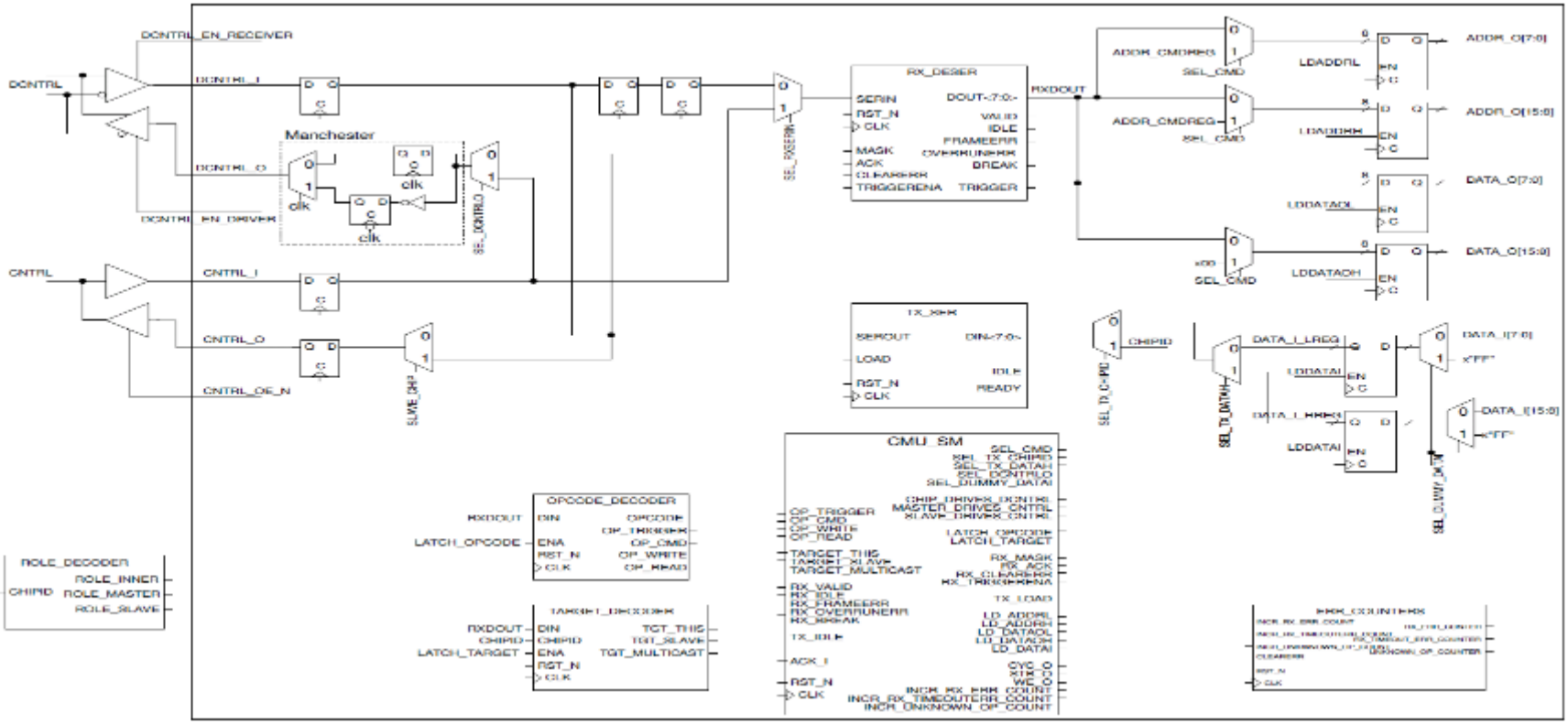
Cons:
- Only applicable to on block level due to state space size

Computational Challenges:
- Memory Limited

PROPERTIES

Problem to solve via Formal

- Verify that CMU returns to IDLE state given a random wake-up state following a power-on? Fail Safe by design?
- FSM with 32 states, encoded 0 to 31. IDLE == 5'd0



Formal Example

- Will CMU return to IDLE state from any start state?

Inputs:
- External control fixed at '1' (IDLE)
- No RST, just CLK
- Register inputs fixed at random state
- Internal variables randomly assigned

```
property reset_not_asserted();
   ( rst_n == 3'b111 );
endproperty

property input_line_dctrl_not_asserted();
   ( dctrl_i == 1 );
endproperty

property input_line_ctrl_not_asserted();
   ( ctrl_i == 1 );
endproperty


//Assumptions
assume_reset: assume property(reset_not_asserted());
assume_dctrl: assume property(input_line_dctrl_not_asserted());
assume_ctrl: assume property(input_line_ctrl_not_asserted());
```

```
property state_goes_to_idle();
   (state_o != IDLE_STATE) |=> (state_o == IDLE_STATE)[->1];
endproperty



// Assertions
assert_state_returns_idle: assert property(state_goes_to_idle());
```

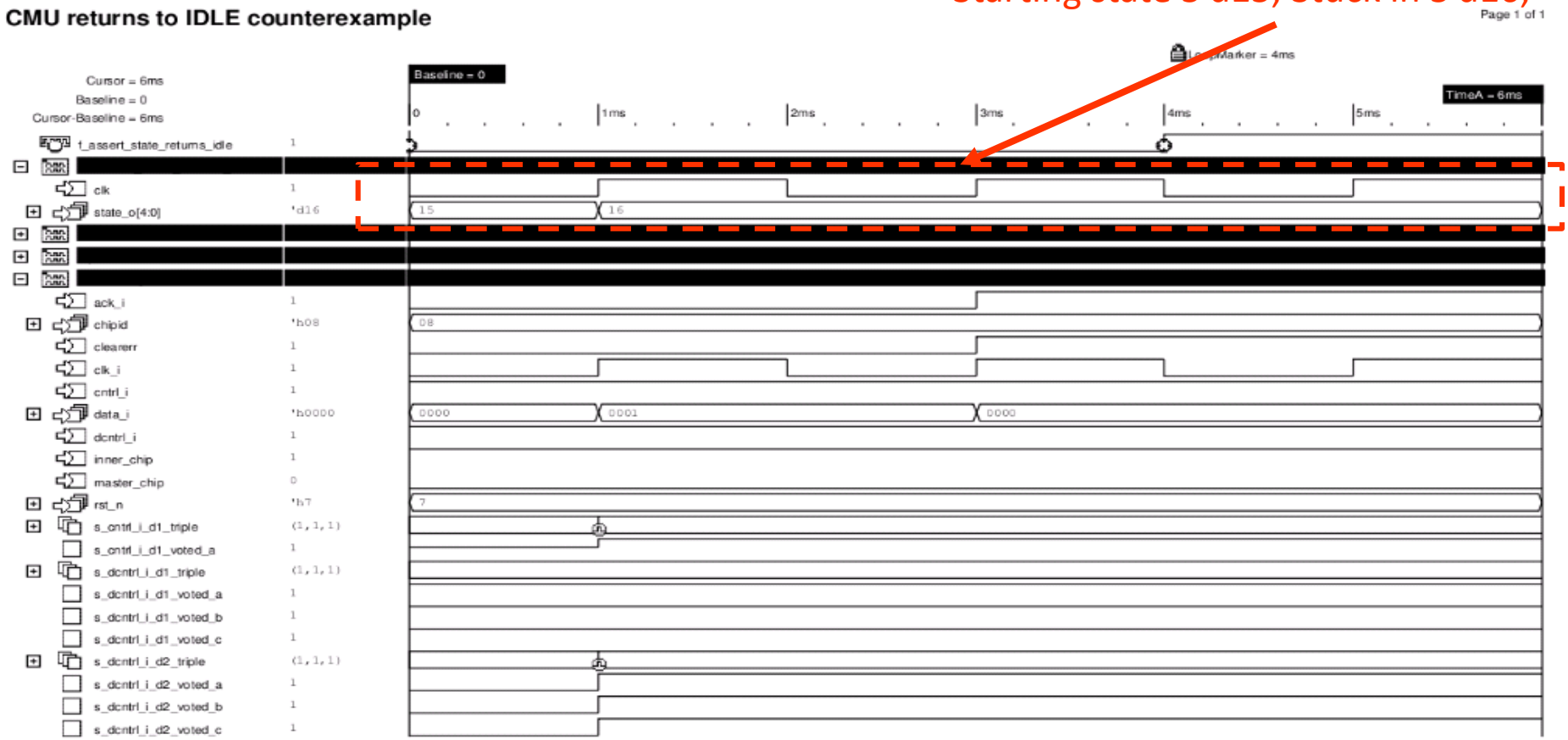
NOTE: Have not initialised a starting state, give full control to tool!

Note: full Formal Control script available in back-up slides

Counter Example

- Will CMU return to IDLE state from any start state?

Starting state 5'd15, Stuck in 5'd16,



=> Answer is NO

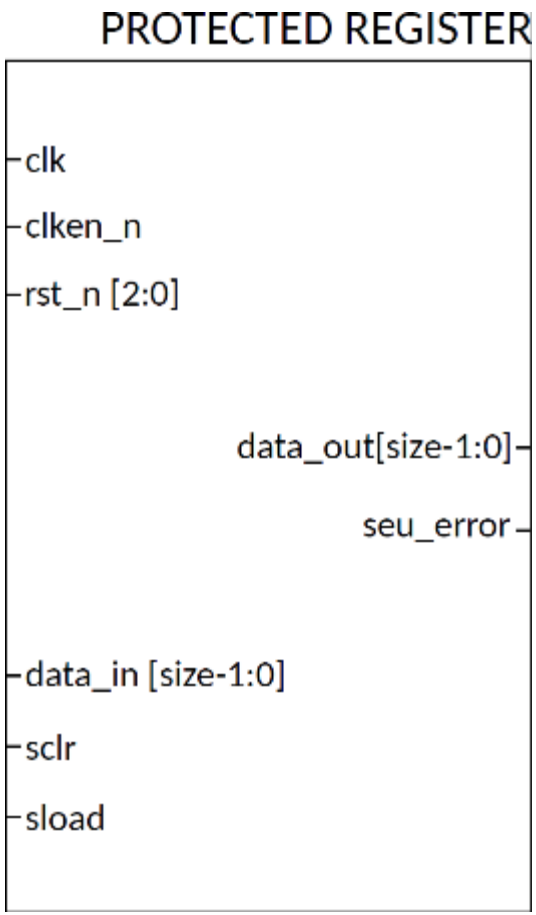
ALPIDE SEU PROTECTION – VERIFICATION

ALPIDE Approach to SEU Protection

- Use a common module “protected register” for any register
- Used in all blocks
- Module preserved, incl. name

Parametrised with:
- Reset Value
- Width
- Type Clock Gating

Protection through TMR



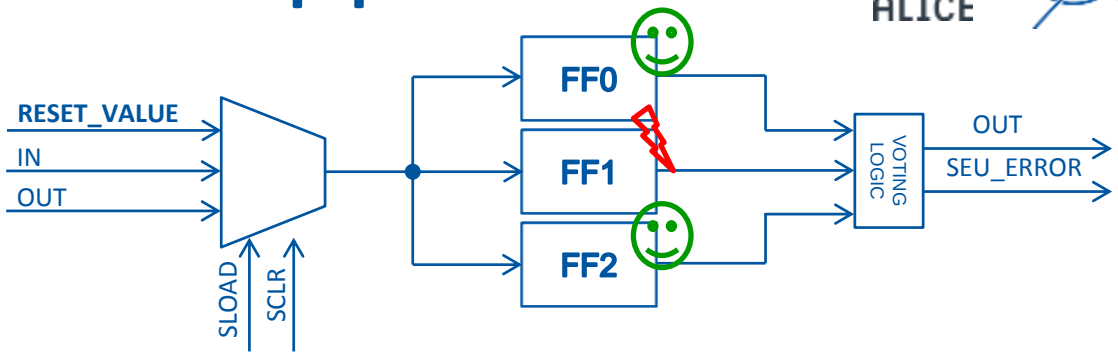
SEU Verification Problem

- 9517 protected bits through TMR

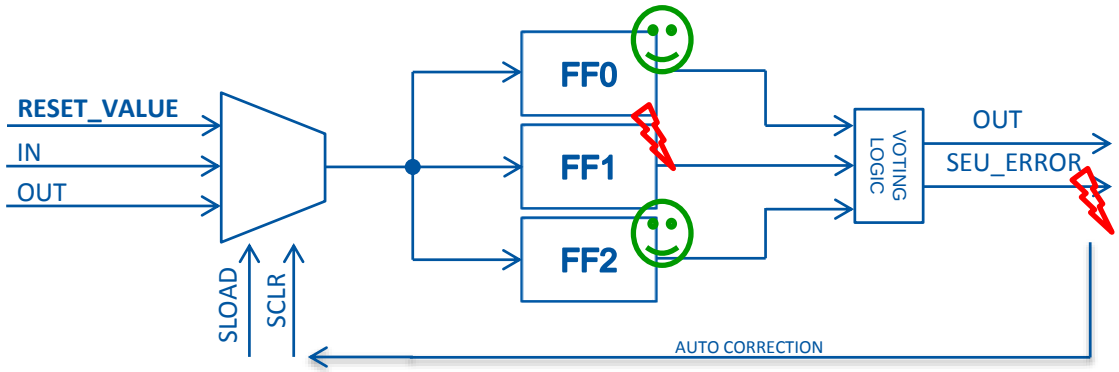
- Need to ensure 3 x 9517 FFs and supporting error correction logic are not simplified following implementation

SEU Verification Approach

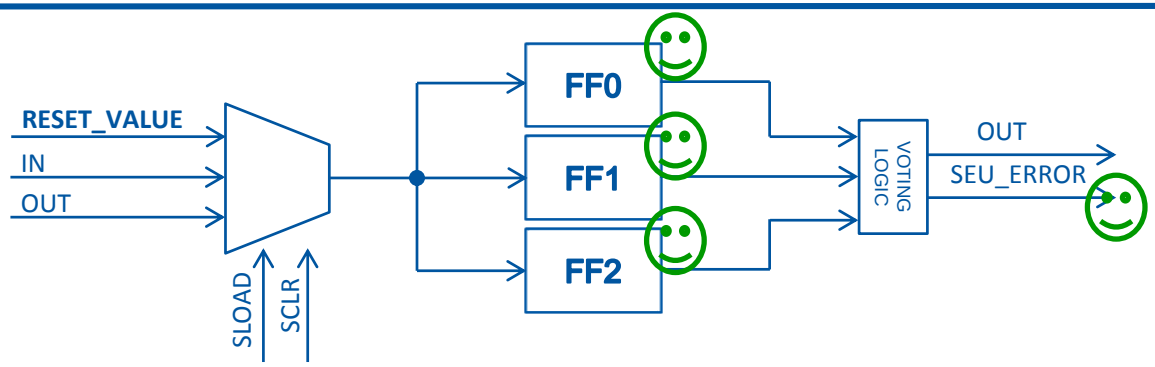
Insert an SEU



Detect and Correct SEU



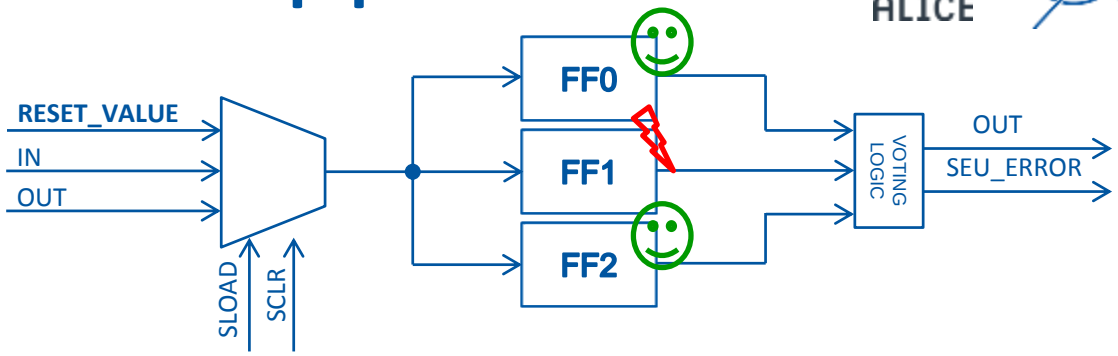
Verify correction



SEU Verification Approach

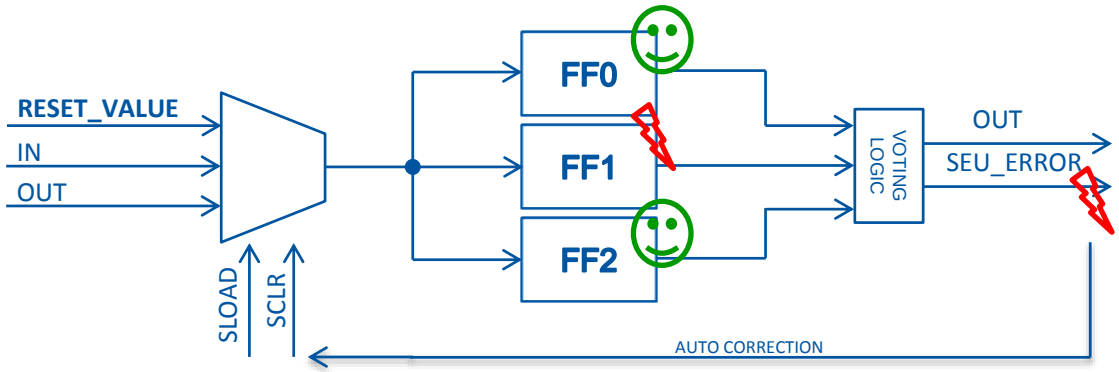
TB

Insert an SEU



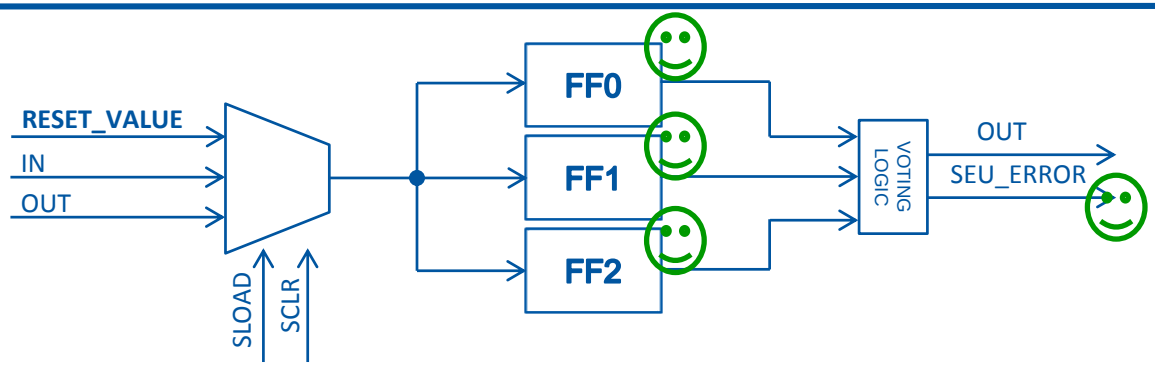
Logic

Detect and Correct SEU



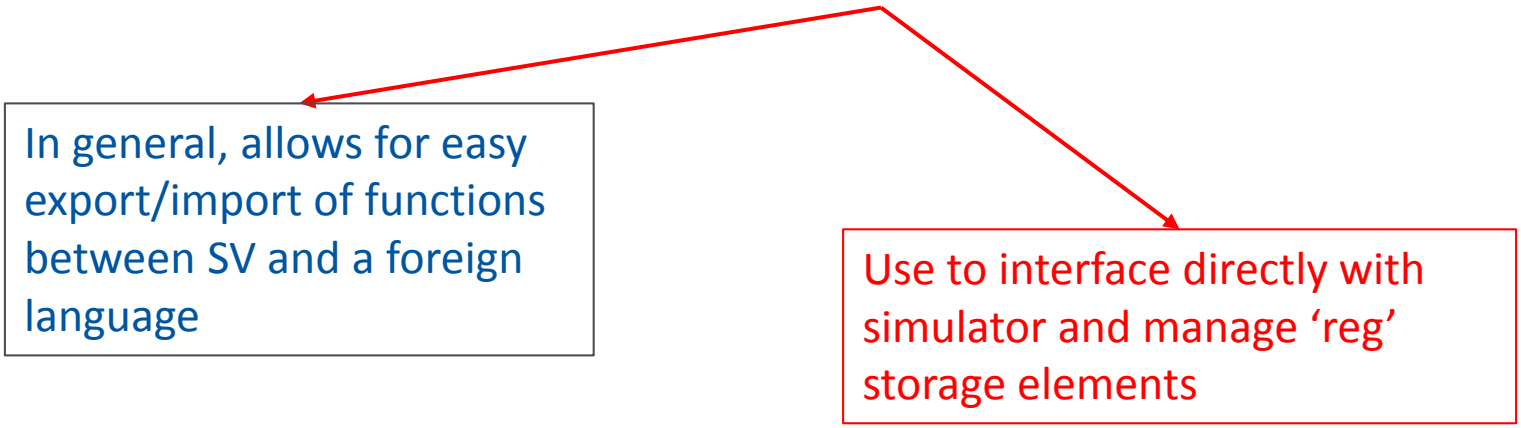
TB

Verify correction



Backdoor access to READ/WRITE required

- FFs in some libraries have a virtual pin to toggle their state (TowerJazz does not)
- Looking for a general approach for any process lib
- = > Use System Verilog Direct Programming Interface (DPI)



UVM backdoor access routines

uvm_hdl_read()

```
import "DPI-C" function int uvm_hdl_read(       string         path,
                                         output uvm_hdl_data_t value)
```

Gets the value at the given *path*. Returns 1 if the call succeeded, 0 otherwise.

uvm_hdl_deposit

```
import "DPI-C" function int uvm_hdl_deposit(string         path,
                                            uvm_hdl_data_t value)
```

Sets the given HDL *path* to the specified *value*. Returns 1 if the call succeeded, 0 otherwise.

uvm_hdl_force

```
import "DPI-C" function int uvm_hdl_force(string         path,
                                          uvm_hdl_data_t value)
```

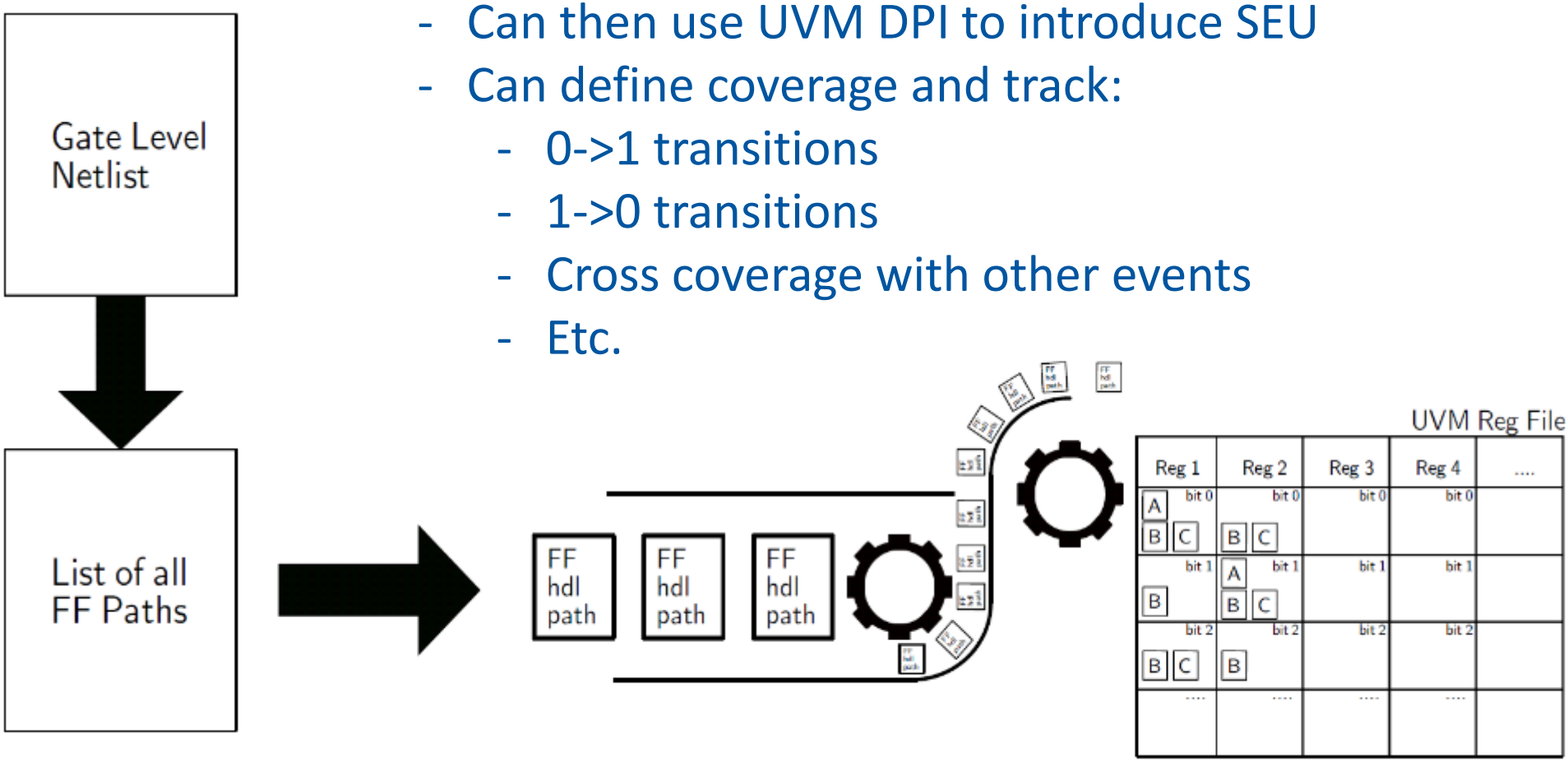
Forces the *value* on the given *path*. Returns 1 if the call succeeded, 0 otherwise.

1. Need only an HDL PATH
2. Fit with UVM Reg File
   - Keep Coverage

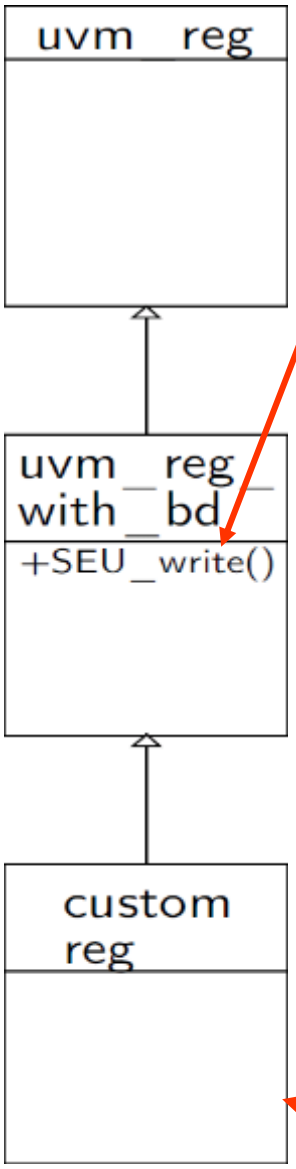
Available through UVM library

Python Parser of HDL Paths

- Sort all FF paths into Register Objects
- Can then use UVM DPI to introduce SEU
- Can define coverage and track:
  - 0->1 transitions
  - 1->0 transitions
  - Cross coverage with other events
  - Etc.



Register Class Structure



SEU_write task:

1. Samples default state of FF via uvm_hdl_read
2. Takes pattern read(e.g. 3'b101) to be the '0' bit state*
3. Toggle **slice A** through via uvm_hdl_deposit
4. Wait 3 clock cycles for error correction
5. Check that bit is restored through uvm_hdl_read
6. Check that SEU count is incremented
7. Repeat steps 3-6 for **slice B** and **slice C**
8. Measure 0->1 coverage for given bit

* Abstraction introduced as details of implementation are not known

** Can later toggle all 3 slices in order to repeat procedure for '1' bit state

Define further coverage/methods as required

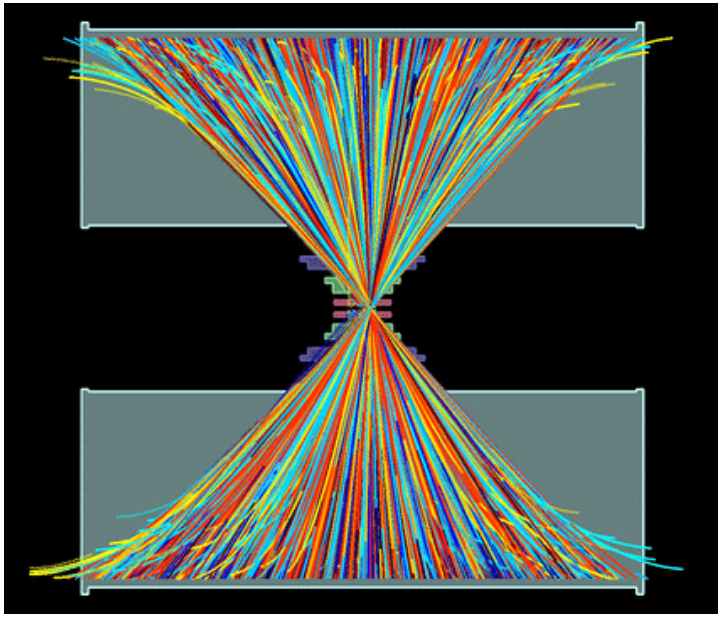
Summary of Approach

Pros	Cons
Runs concurrently with simulations	Requires registers names to be preserved in implementation
Coverage collection	Additional complexity
Integration with existing UVM Reg File structure	
Not library specific	

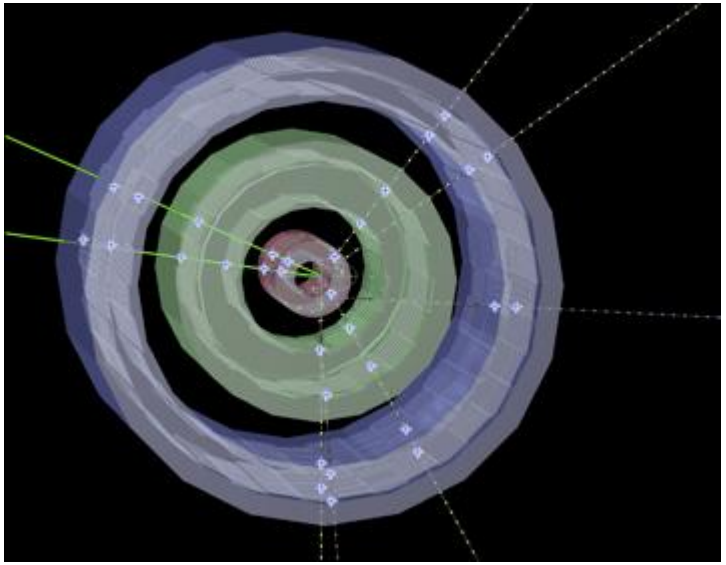
Overall Summary

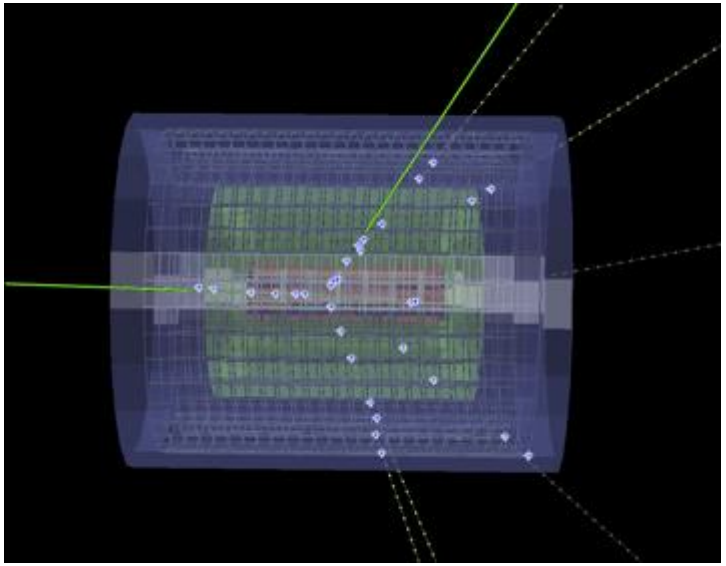
- ALPIDE chip: 90M transistors,20k gates
- SV-UVM constrained random and metric driven verification approach
- Incremental Snapshot Elaboration can boost simulation verification efforts
- ABV/Formal techniques should be used in parallel with simulation
- SEU Verification can be facilitated via UVM DPI

BACK-UP SLIDES

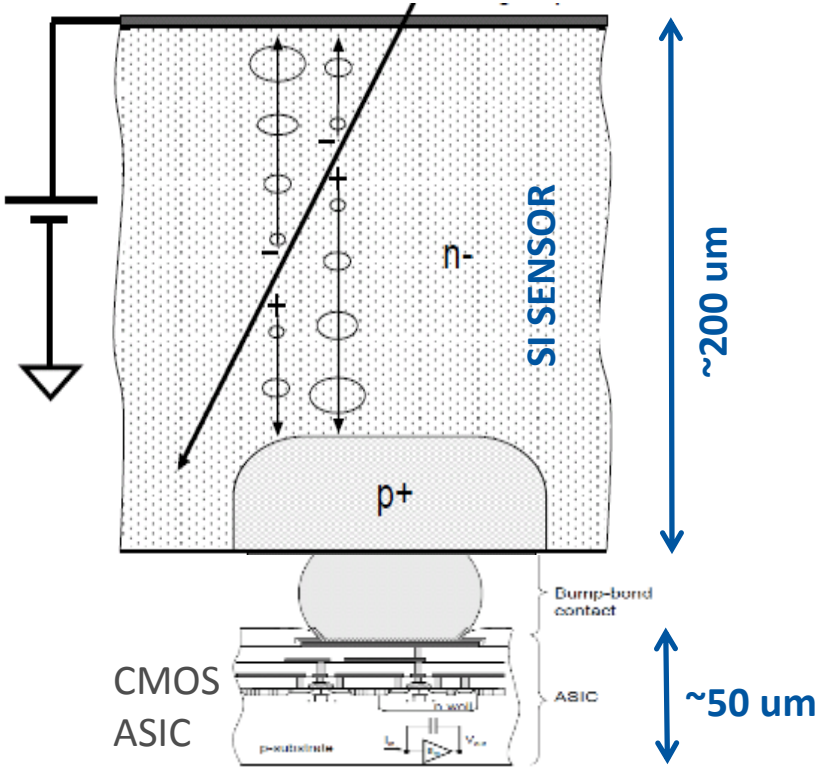


LHC Pb-Pb collision (ALICE, Sep 2011)





Hybrid Pixels



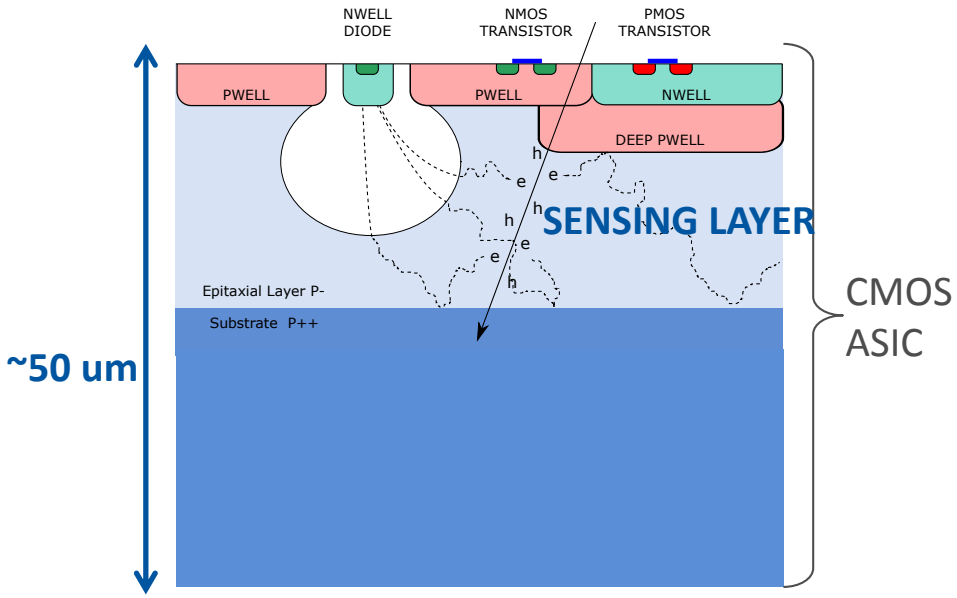
Widely employed in today's detectors:

ALICE, ATLAS, CMS, LHCb (RICH), NA62 GTK, …

Applications in several other fields

MEDIPIX, TIMEPIX, …

Monolithic Pixels



Current application in HEP: vertex detector of STAR experiment

Choice for the Upgrade of the ALICE ITS

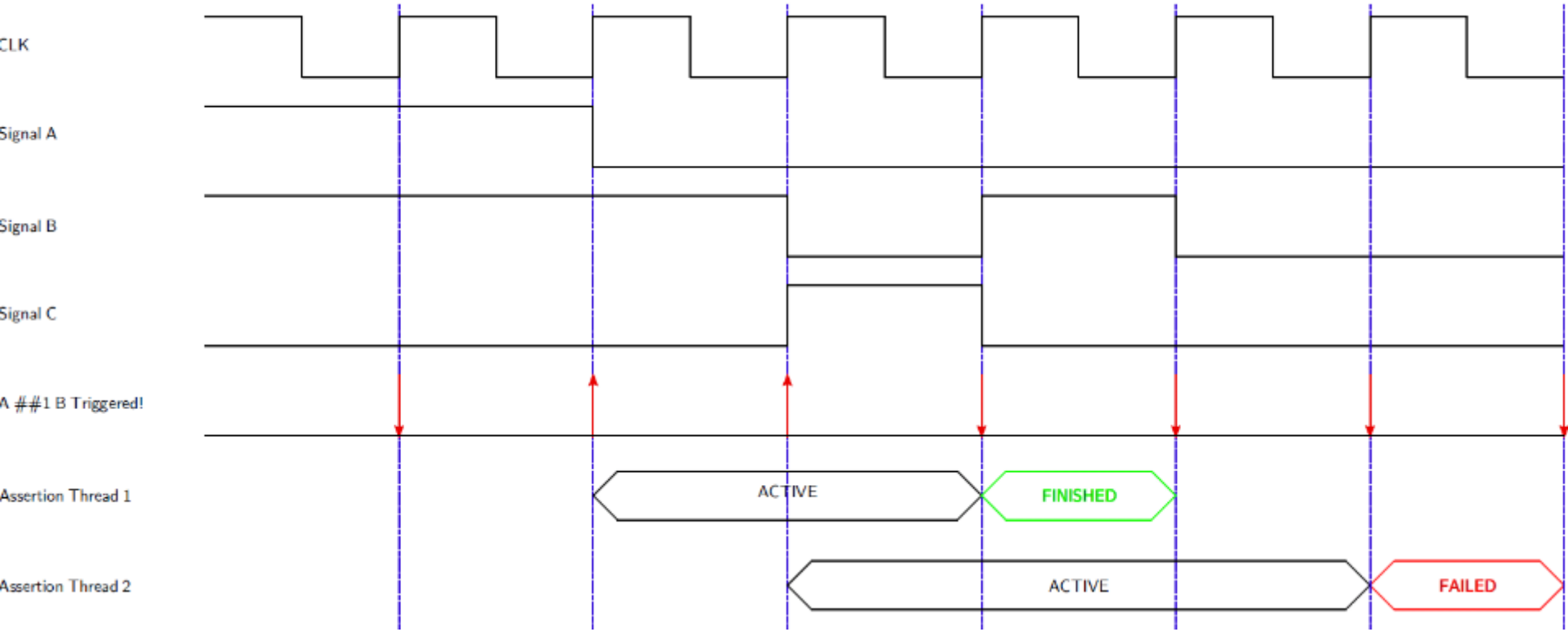
Major leaps forward in technology, architecture and design

Temporal Sequences – Example
Spawning of Sequences

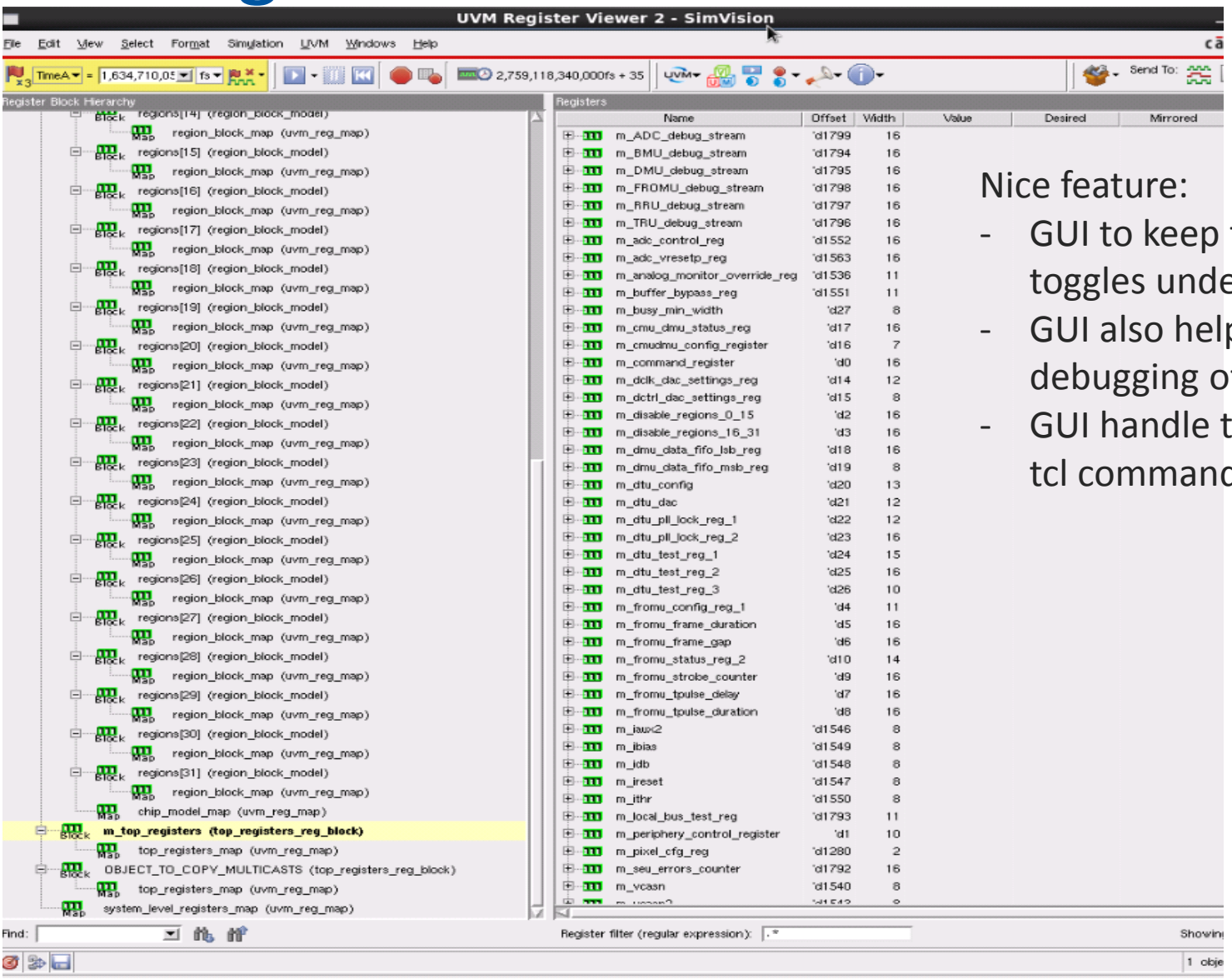
A ##1 B |=> (B ##1 C)[->1]

trigger sequence

asserted sequence



SEU Verification – Integration with UVM



Nice feature:

- GUI to keep track of inserted toggles under desired column
- GUI also helpful for general debugging of sim
- GUI handle to introduce SEU via tcl commands