Emulation Moves Into 4-State Logic and Real Number Modeling

DVCon US 2024

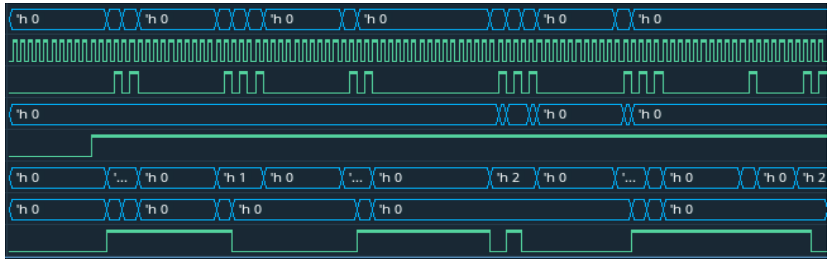
Brad Budlong, Michael Young - Cadence Design Systems
Kyoungmin Park - Samsung Semiconductor
Nimay Shah - Analog Devices

2024 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES
SAN JOSE, CA, USA
MARCH 4-7, 2024

cādence®

New Emulation Processor Expands Emulation Applications

Traditional 2-State Emulation

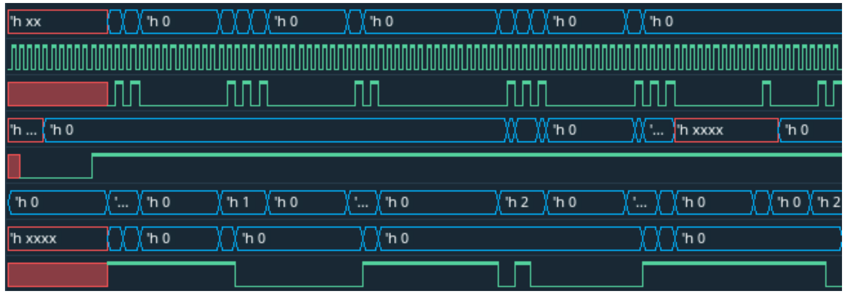


- Long history of increasing capacity and performance
- Implemented with Emulation Processors and FPGAs

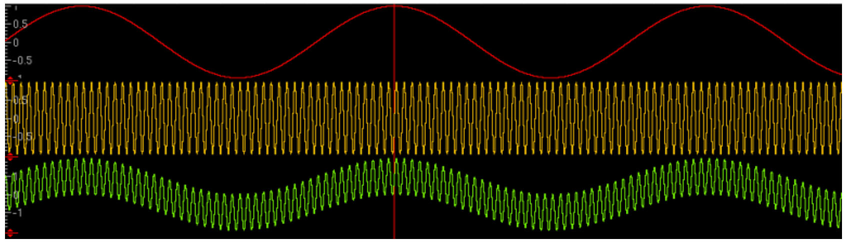
4-State

Real Numbers

4-State Emulation



Real Number Modeling (RNM)



Palladium Z2 Emulation Processor
- Increased capacity and performance
- Added support for 4-state and real number modeling

4-State Emulation

Benefit of 4-State in Emulation

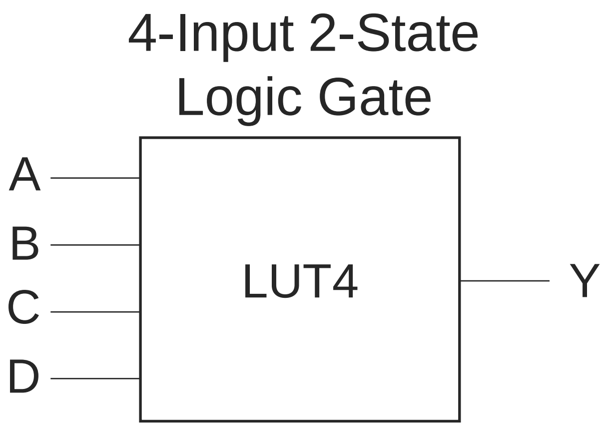
- Power aware (UPF) verification of large devices
  - Power sequences that are too long and complex for simulation
  - Corruption with 2-state random values is difficult to trace to the origin
  - 2-state verification is not sufficient for low power signoff

- Lengthy initialization sequences
  - Multiple phases executed to complete the overall initialization sequence

- Extended tests sensitive to out of bounds / uninitialized values
  - X values can find errors that would otherwise be missed

Brute Force 4-State Emulation Logic Implementation

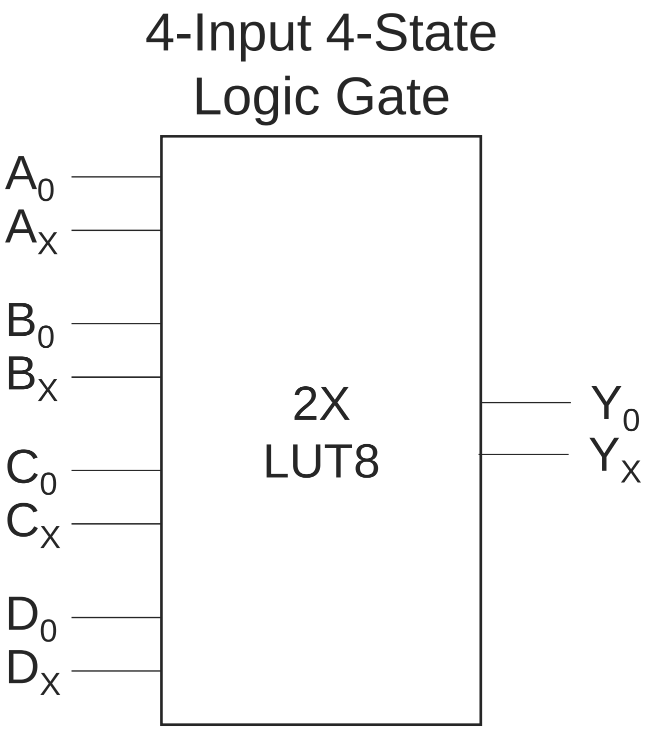
Standard 2-State Implementation



$2^4 = 16$ bits

32x Increase

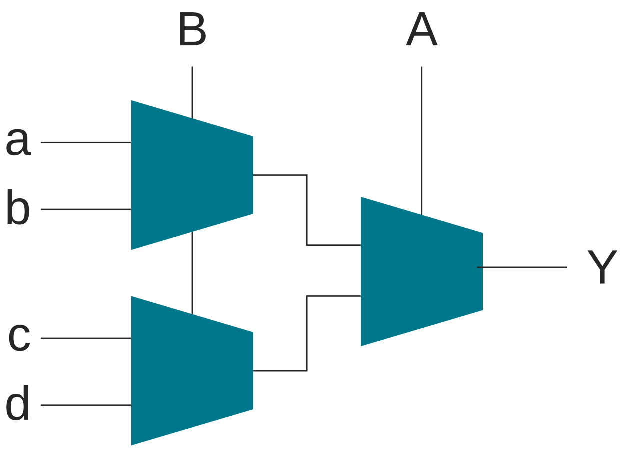
Brute Force 4-State Implementation

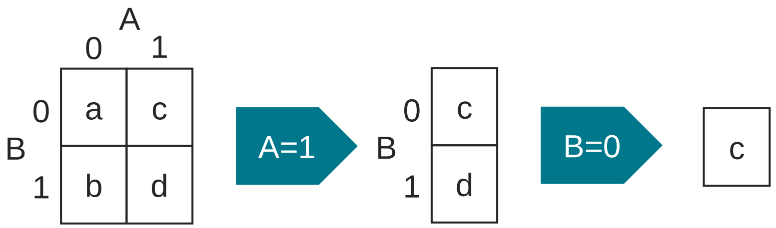


$2 * 2^8 = 512$ bits

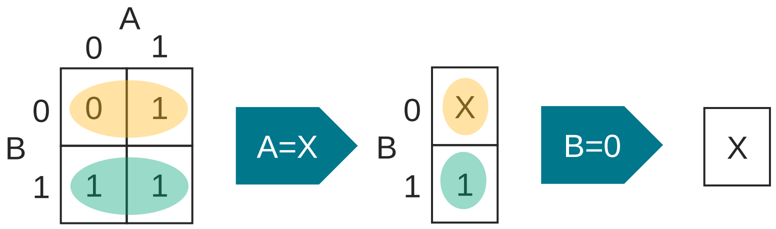
LUT as a Chain of Muxes

LUT2 Implemented with Muxes





LUT2 Implemented with 4-State

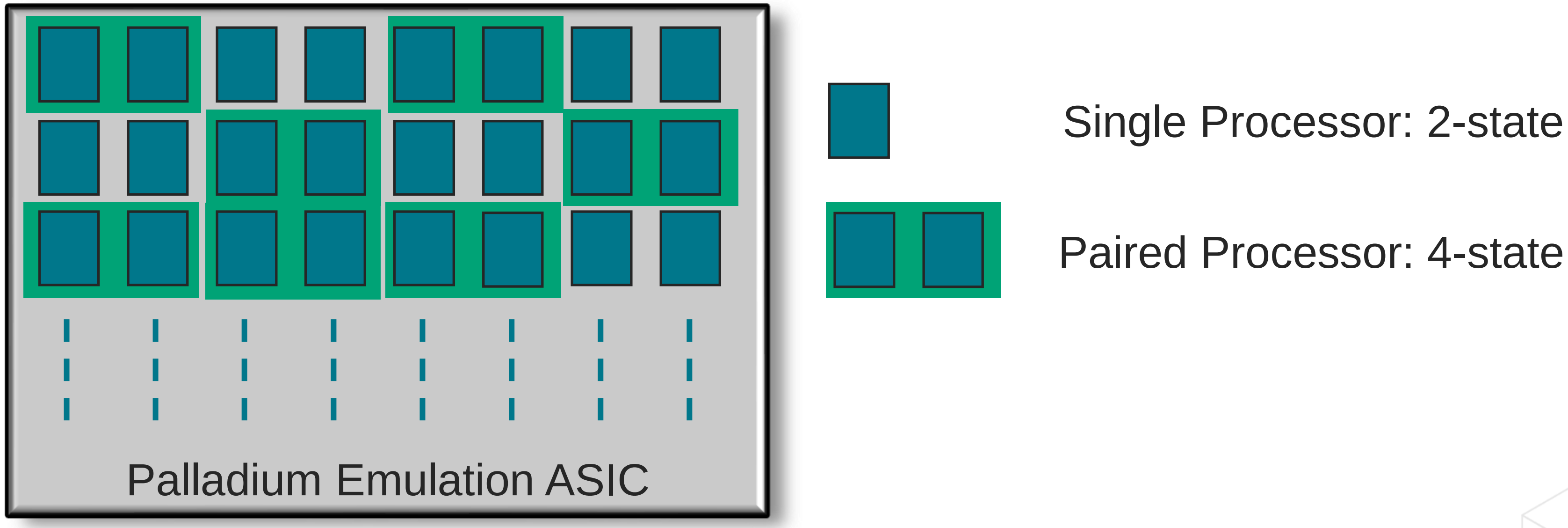


Compute As Ternary (CAT) mode calculation

For each input:
- 0/1: Select half the entries
- X: Result is 0/1 if matching, X if different

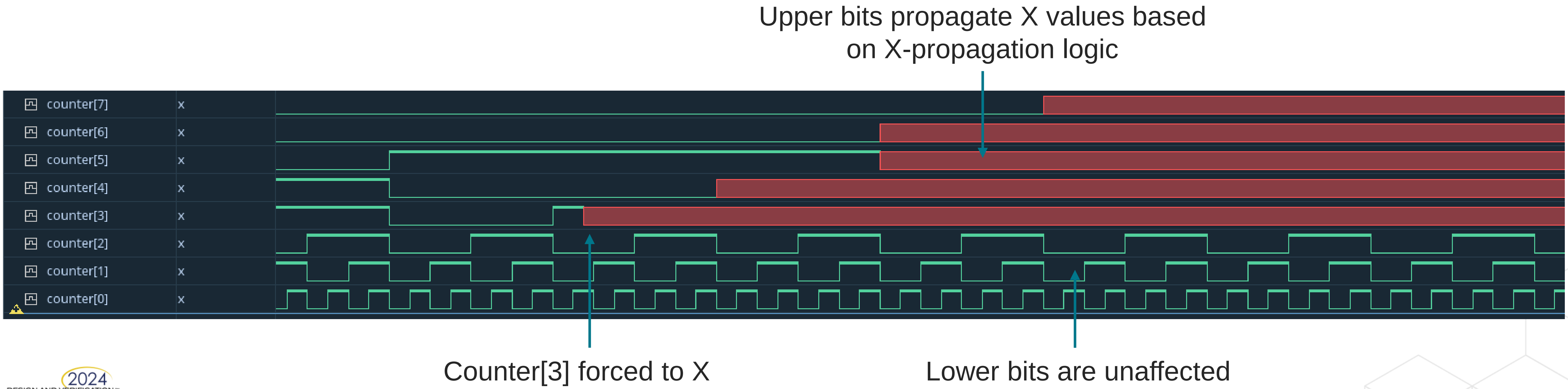
Practical 4-State Implementation with Processor Based Emulation

- Palladium Z2 introduces processors that can be paired to implement 4-state operation
  - Palladium Z2 has 1,000s of processor per chip
  - Single processors used for 2-state signals
  - Processor pair used for 4-state signals
    - Computation, registers and routing implemented with a processor pair
    - Processor pair operates at the same clock frequency as single 2-state processors
  - Overall capacity cost of ~2.3x for 4-state versus 2-state
    - Increased capacity only for the portion of the design compiled for 4-state



X-Handling Implementation

- X-handling is implemented at the processor granularity
  - X computation within a processor pair with up to 4 inputs done in CAT mode
- Calculations done from a synthesized netlist
  - This is the same as all emulation where the design is first synthesized before mapping to the emulator
- Yields X-pessimism closer to hardware operation than Verilog LRM handling
  - Similar to gate level X handling but with less X-pessimism since computation is done on larger blocks of logic



4-State Handling of State Elements

- RTL and Gate representation is unchanged for 4-state
  - Still emulating the same design as 2-state
  - X handling is done in the implementation and not written into the RTL

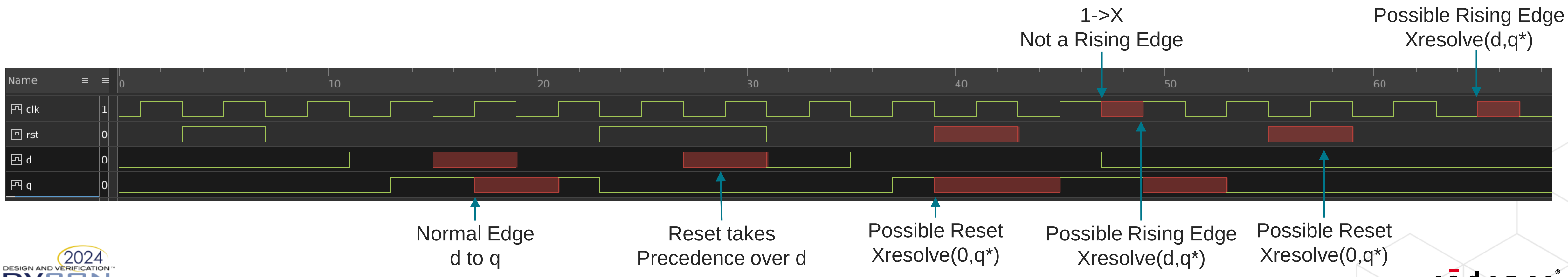
Flip-Flop with Asynchronous Reset
Code Template

```
always_ff @(posedge clk or posedge rst) begin
  if (rst) begin
    q <= 1'b0;
  end else begin
    q <= d;
  end
end
```

Flip-Flop with Asynchronous Reset
4-State CAT Mode Behavior

rst	clk	q
1	Don’t Care	0
0	0->1	d
0	0->X or X->1	Xresolve(d,q*)
X	No edge	Xresolve(0,q*)

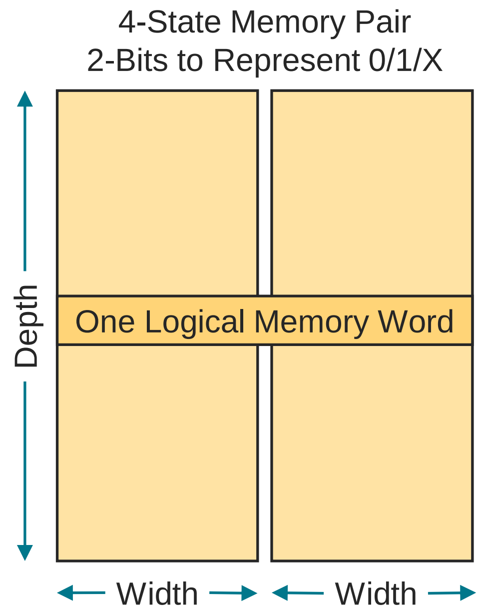
q* is the current q value. Xresolve(a,b) := a if a === b else X

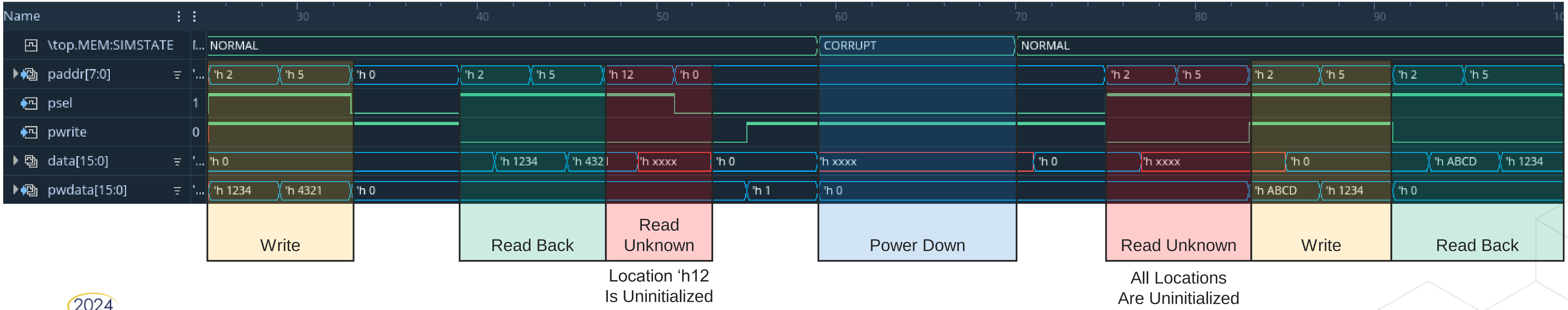


cādence®

4-State Handling of Memories

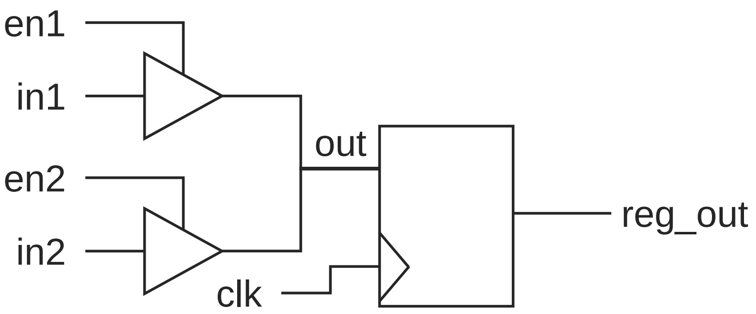
- RTL representation is unchanged for 4-state memories
  - A pair of physical memories are used to implement the logical 4-state memory
  - 2-bits used for the 3 encoded memory data values (0/1/X)
- Instantaneous complete memory corruption is logically implemented
  - UPF corruption event in the power domain of the memory
  - Write with an X address





Implementing the Z State

- 2-State emulation wired OR implementation misses functional error conditions
- Z state can be added with a logical transformation
  - Only used on a small number of signals
  - Hardware X support is needed for downstream logic

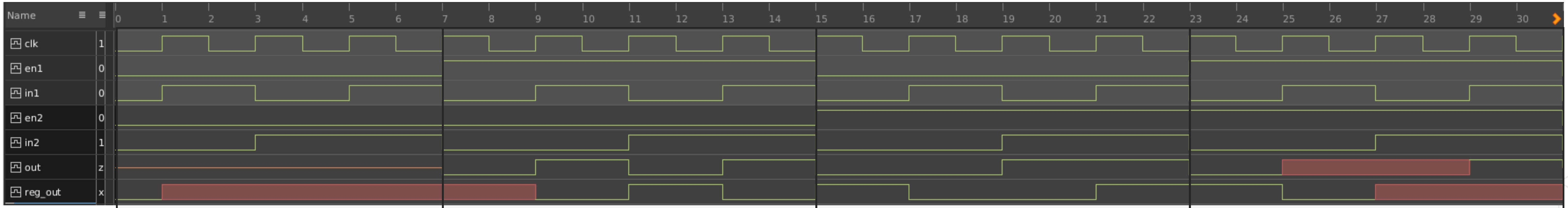


2-State Wired OR Implementation

Driver 1	Driver 2	Result
0	0	0
0	1	1
0	Z	0
1	0	1
1	1	1
1	Z	1
Z	0	0
Z	1	1
Z	Z	0

4-State X/Z Implementation

Driver 1	Driver 2	Result
0	0	0
0	1	X
0	Z	0
1	0	X
1	1	1
1	Z	1
Z	0	0
Z	1	1
Z	Z	Z



No Driver	Driver 1	Driver 2	Both Drivers
out = Z, reg_out = X	out = in1	out = in2	out = X when in1 ≠ in2

4-State Throughout the Emulation Flow

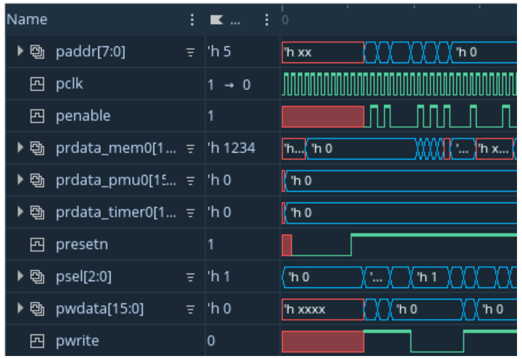
Force and Value

Support of 0/1/X

```
XErun> value timer0.counter
16'b0000000000110010
XErun> force timer0.counter[2] 1'bx
XErun> run 20ns
Ran until 720 NS + 0
XErun> value timer0.counter
16'b00000000xxxxxx00
```

Waveforms

Emulation waveform reconstruction including X/Z



Memory Load and Dump

0/1/X values in all the memory file formats

```
$INSTANCE top.mem0.memarr
$RADIX   HEX
$ADDRESS   0    ff
0  xxxx
1  xxxx
2  1234
3  xxxx
4  xxxx
5  4321
6  xxxx
7  xxxx
```

DPI-C

4-state values with standard svLogicVecVal C type

```
void print4s(svLogicVecVal *val) {
  printf("4-State cnt: ");
  for (int i = 7; i >= 0; i--) {
    if ((1 << i) & val->bval) {
      printf("X");
    } else  {
      if ((1 << i) & val->aval) printf("1");
      else printf("0");
    }
  }
  printf("\n");
}
```

Assertions

Synthesis support for === and !== with X and Z constants along with $isunknown

```
valid_we: assert property (@(posedge pclk) !psel || (pwrite !== 1'bx))
else $warning ("X on Memory Write Enable");
```

SAMSUNG

2024 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES
SAN JOSE, CA, USA
MARCH 4-7, 2024

4-State Logic Emulation

DVCon US 2024

Kyoungmin Park

Principal Engineer, SOC Design Verification team

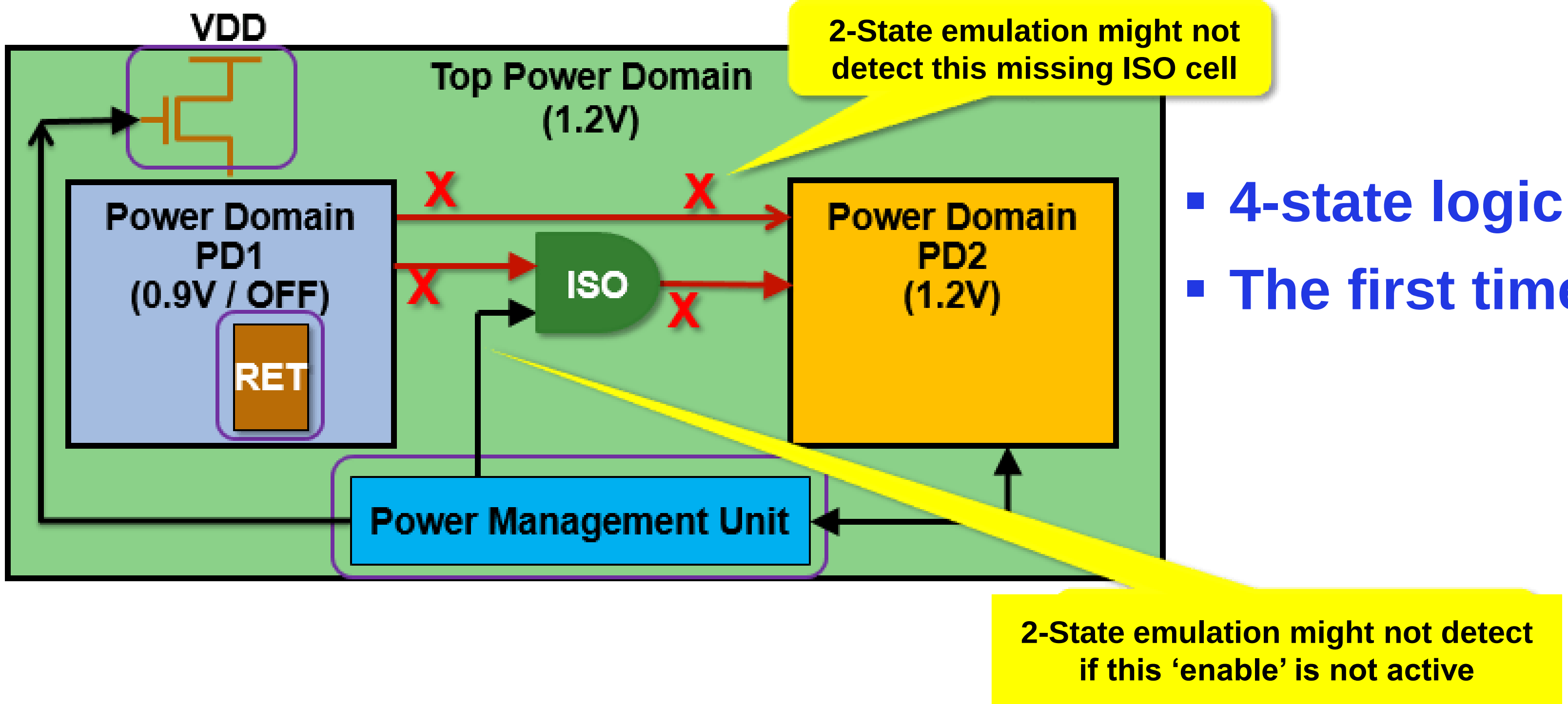
Samsung Semiconductor

March 7, 2024

Business Portfolio

4-State Logic Emulation

- **Background**
  - Power aware verification including UPF is always a long pole in SOC projects.
  - Traditional emulation supports only 2-state logic (‘0’ and ‘1’). ‘X’ and ‘Z’ are not supported.
  - We might miss power intent bugs.



- **4-state logic emulation is enabled**
- **The first time in this industry**

4-State Logic Emulation: Resource Analysis

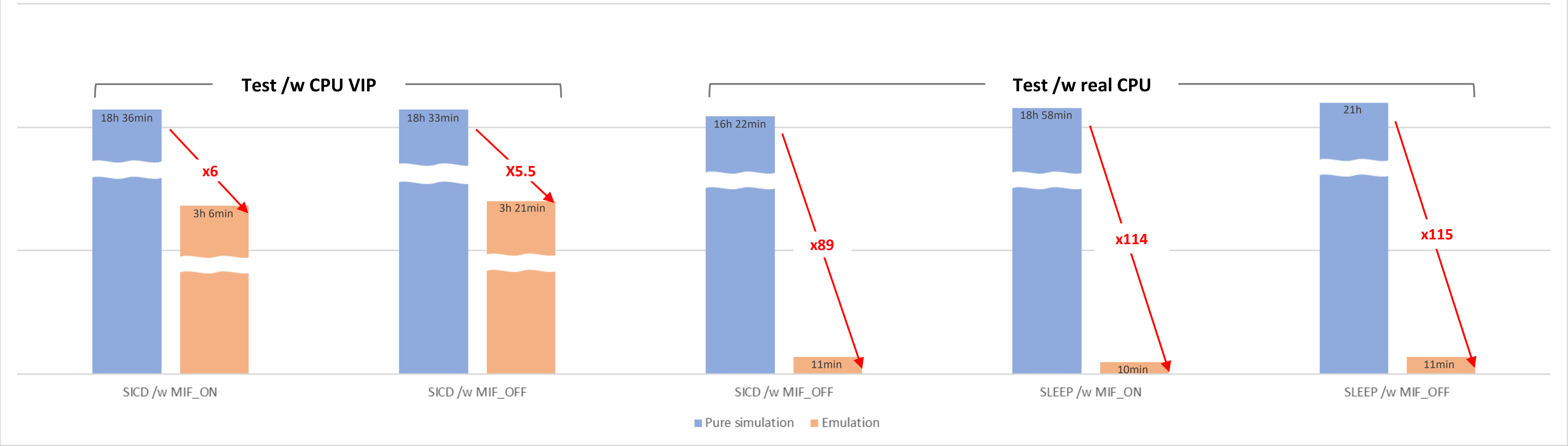
- 4-State requires approx. **2.3 times more resources** than 2-State emulation

Mode	Relative Cost	2-State w/o UPF	2-State w/ UPF	4-State w/o UPF	4-State w/ UPF
Emulation capacity	vs. 2-State	1	1.4x	2.3x	3.2x
	vs. 2-State w/ UPF		1		2.3x
Compile time	vs. 2-State	1	1.5x	1.5x	2.1x
	vs. 2-State w/ UPF		1		1.4x
Performance	vs. 2-State	1	0.75x	0.9x	0.59x
	vs. 2-State w/ UPF		1		0.81x

NOTE. ‘Emulation capacity’ refers to the volume of resources utilized by the emulator hardware

4-State Logic Emulation: Results

- **4-State performance comparison between Pure simulation and Emulation**



NOTE. Because there is almost no interface b/w H/W(DUT) and S/W(TB) in real CPU env.,
the performance improvement of test /w real CPU env. is much higher than that of test /w CPU VIP env.

Real Number Modeling (RNM) Emulation

Benefit of RNM in Emulation

- Large mixed signal ASICs
  - Complete mixed signal design is too large for significant simulation
  - Emulation didn’t support RNM, so the analog had to be black boxed
  - Now a mixed signal emulation is possible, enabling the interaction between digital and analog to be tested over a longer window

- Memory and Serdes interfaces in otherwise digital designs
  - Most digital designs include physical interfaces that have analog characteristics
  - Now these physical interfaces can be emulated along with their digital control

- Large scale real number applications
  - Emulation enables higher performance for applications with large amounts of real number calculations

Palladium Z2 Real Number Modeling (RNM)

- Hardware support for SystemVerilog real data types and real operations
- Transformation of SystemVerilog user defined nettypes (UDN) to synthesizable logic
- Addition of efficient time delay support commonly used in RNM designs

Hardware Support for Real Number Operations

- Palladium Z2 includes Floating Point Units (FPU)
  - Each chip has 64 FPUs available
  - FPUs are time division multiplexed
    - Allows each FPU to execute 10’s of operations
    - 1 Billion Gate Z2 emulator can execute >10K FPU operations
- RNM support includes direct support for real data types
  - real (64-bit), shortreal (32-bit) and realtime (64-bit)
    - Some operations are only supported with shortreal precision
  - Unpacked struct, arrays, and combinations with real data type components
  - Support for all SystemVerilog math operators and math system functions

Operator	Description	Data Types
=	Assignment	real, shortreal
+= -= /= *=	Arithmetic assignment	real, shortreal
?:	Conditional	real, shortreal
+ -	Unary arithmetic	real, shortreal
!	Unary logical	real, shortreal
+ - * /	Binary arithmetic	real, shortreal
**	Power	shortreal
&& || -> <->	Binary logical	real, shortreal
< <= > >=	Binary relational	real, shortreal
== !=	Binary equality	real, shortreal
++ --	Unary increment / decrement	real, shortreal

Function	Description	Data Types
$ln(x)	Natural log	shortreal
$log10(x)	Log base 10	shortreal
$exp(x)	Exponential (e^x)	shortreal
$sqrt(x)	Square root	real, shortreal
$pow(x,y)	Power	shortreal
$floor(x)	Floor	real, shortreal
$ceil(x)	Ceiling	real, shortreal
$sin(x)	Sine	shortreal
$cos(x)	Cosine	shortreal
$tan(x)	Tangent	shortreal
$hypot(x,y)	Hypotenuse sqrt(x^2+y^2)	real, shortreal
$sinh(x)	Hyperbolic sine	shortreal
$cosh(x)	Hyperbolic cosine	shortreal
$tanh(x)	Hyperbolic tangent	shortreal

Example Usage of Real Data Types, Operators and Functions

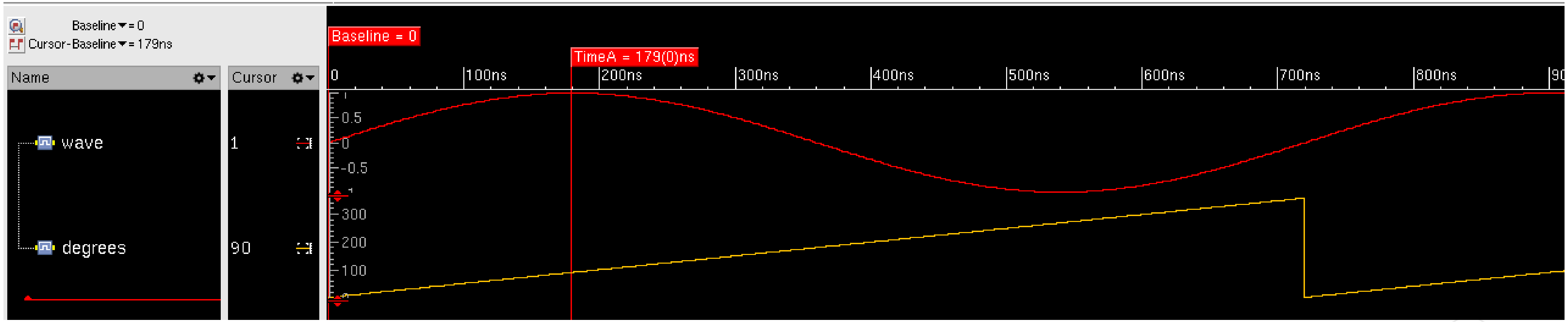
Sine Wave Generator Module

```
module sine_wave #(parameter real step) (
    input clk,
    output real val
);
  const real degrees2rad = 3.14159265 / 180.0;
  real degrees, next_degree;
  always_comb begin
    next_degree = degrees + step;
    if (next_degree >= 360.0) next_degree = next_degree - 360.0;
  end
  always @(posedge clk) begin
    degrees <= next_degree;
  end
  assign val = $sin(degrees * degrees2rad);
endmodule
```

real types used for signals

Operations on reals

Sine function call



2024 DESIGN AND VERIFICATION DVCON CONFERENCE AND EXHIBITION UNITED STATES

cadence

User Defined Nettype (UDN) Support

- User defined nettypes provide custom handling of signal resolution
- Resolution function receives an input dynamic array of the base data type
  - Must be synthesizable when the dynamic array is automatically replaced by a fixed sized array

User Defined Nettype

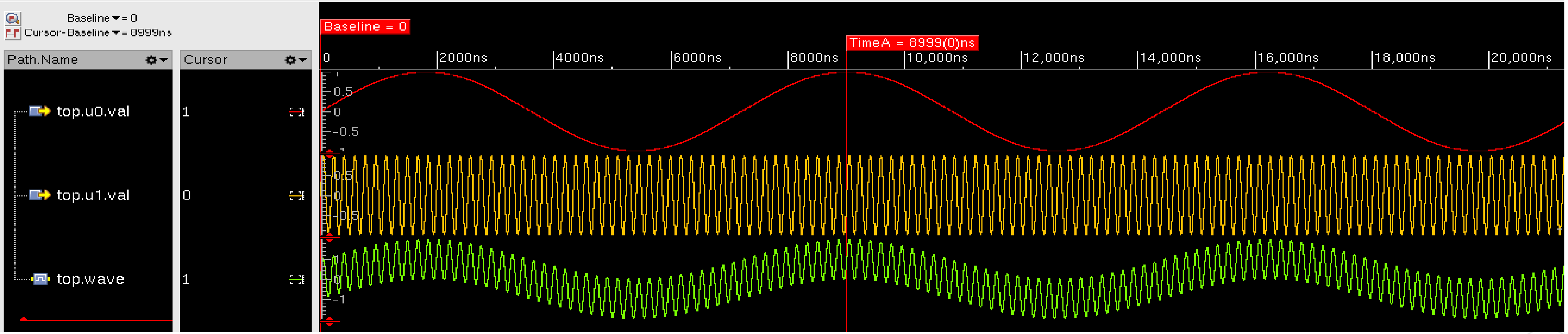
```
function automatic real rsum (input real driver[]);
  foreach (driver[i]) begin
    rsum += driver[i];
  end
endfunction
nettype real real_sum with rsum;
```

Resolution function

nettype declaration

Two Sine Waves Summed

```
real_sum wave;
sine_wave #(0.1) u0 (clk, wave);
sine_wave #(4.0) u1 (clk, wave);
```



Support for Time Delays with RNM

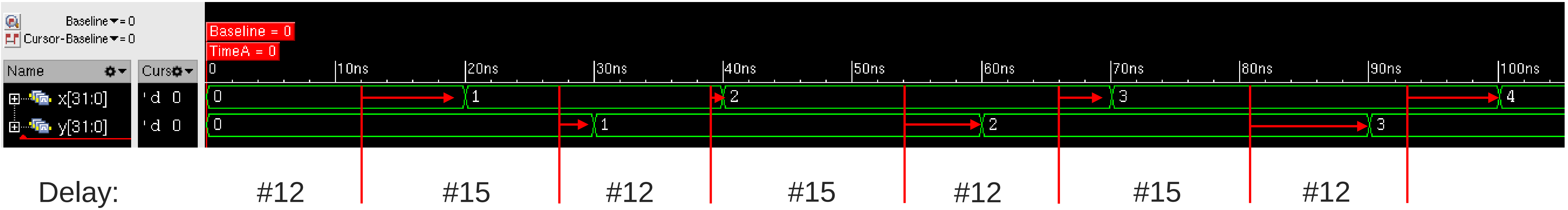
- Time delay control (ie. #5) is not synthesizable
  - Digital emulation makes an exception to handle clocks using specific clocking models
- For RNM, constant and variable time delays are often needed
  - Tuning of signal delays
  - Ordering of logic computation
  - Breaking of combinational loops
- RNM time delay support adds transformations to handle specific constructs
  - always process with time delay and no event
    ```
    always #10 a = b;
    ```
  - always process with time delay and explicit event
    ```
    always @(a or b) begin #10; y = a + b; end
    ```
  - Continuous assignment with a single delay
    ```
    assign #10 y = a + b;
    ```
- Delays can be constants or variables and integer or real
- Functionality is the same as simulation

Alignment of Time Delays with the Edges of the Fastest RTL Clock

- Palladium executes only on edges of the fastest RTL clock in the system
- RNM delays are automatically transformed to align to these clock edges
  - The delay expires at the next edge equal to or after the accumulated delay
  - Subsequent delays take into consideration the accumulated overshoot from previous delays

Always Process with Multiple Delays

```
always begin
  #12 x = x + 1;
  #15 y = y + 1;
end
```



Delay: #12 #15 #12 #15 #12 #15 #12

Accumulated Delay: 12 27 39 54 66 81 93

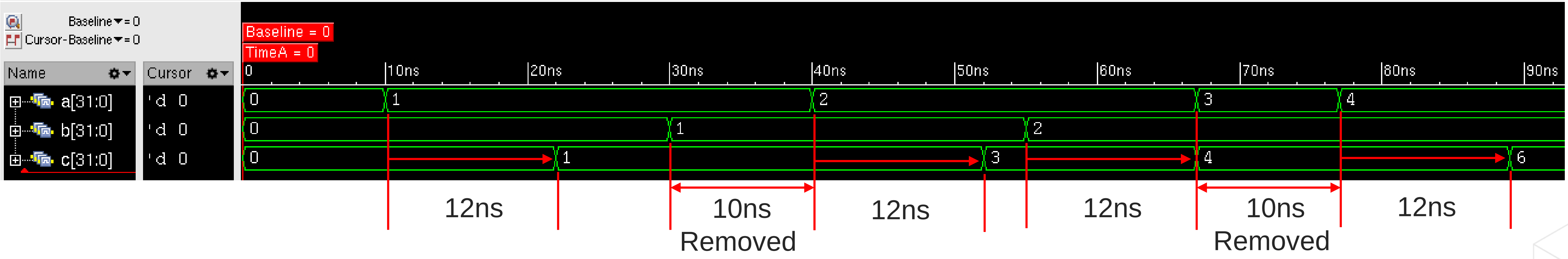
10ns Edges for the Fastest RTL Clock

Support for Continuous Assignment Delay

- Support is added for a single time delay in a continuous assignment
- Modeled as an inertial delay as described in the SystemVerilog standard
  - New value is scheduled for the output after the specified delay
  - If the value changes before the value propagates, that scheduled output is removed and the new value is scheduled

Continuous Assignment Delay

```
assign #12 c = a + b;
```



Scheduled output values of 2 and 5 removed

ANALOG DEVICES

2024 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

SAN JOSE, CA, USA MARCH 4-7, 2024

Enabling True System-Level, Mixed Signal Emulation

Nimay Shah (Presenter), Paul Wright, Pranav Dhayagude

System Verification and Validation
CTO Office

analog.com

Agenda

Analog Devices and the Intelligent Edge

Verification of Digitally Controlled Mixed-Signal Systems

Previous Approaches to Mixed-Signal Emulation

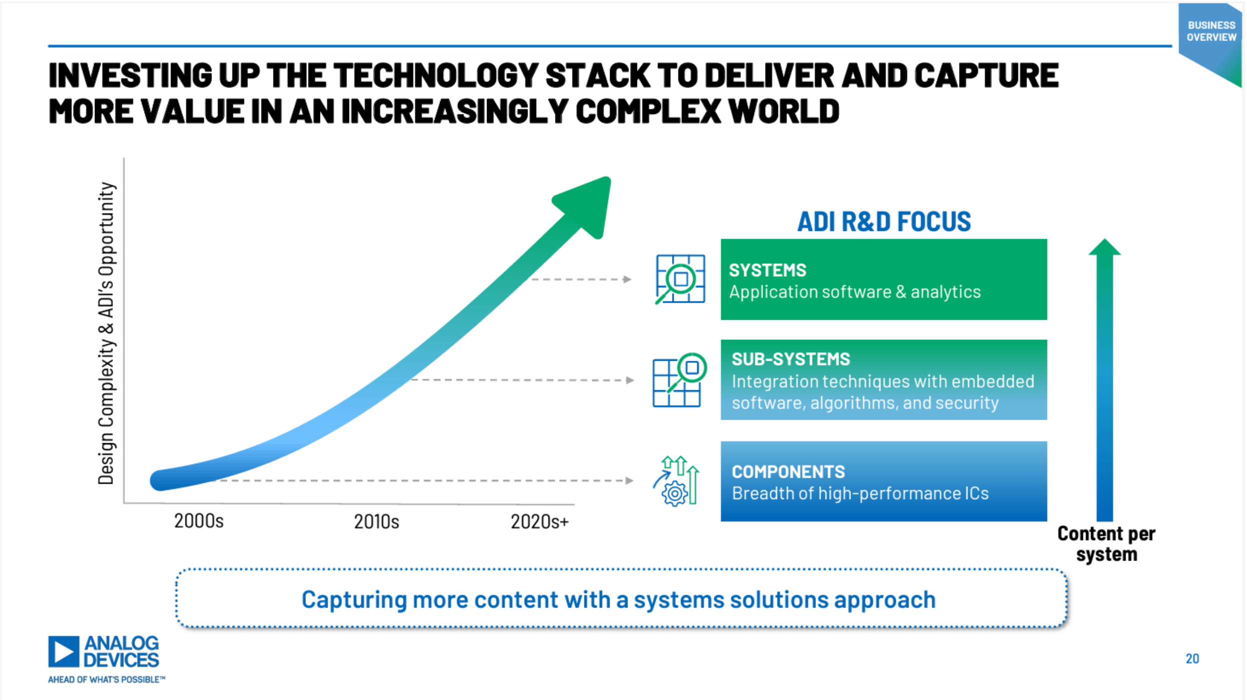
Emulation of SystemVerilog Real Number Models (RNM) on Palladium Z2

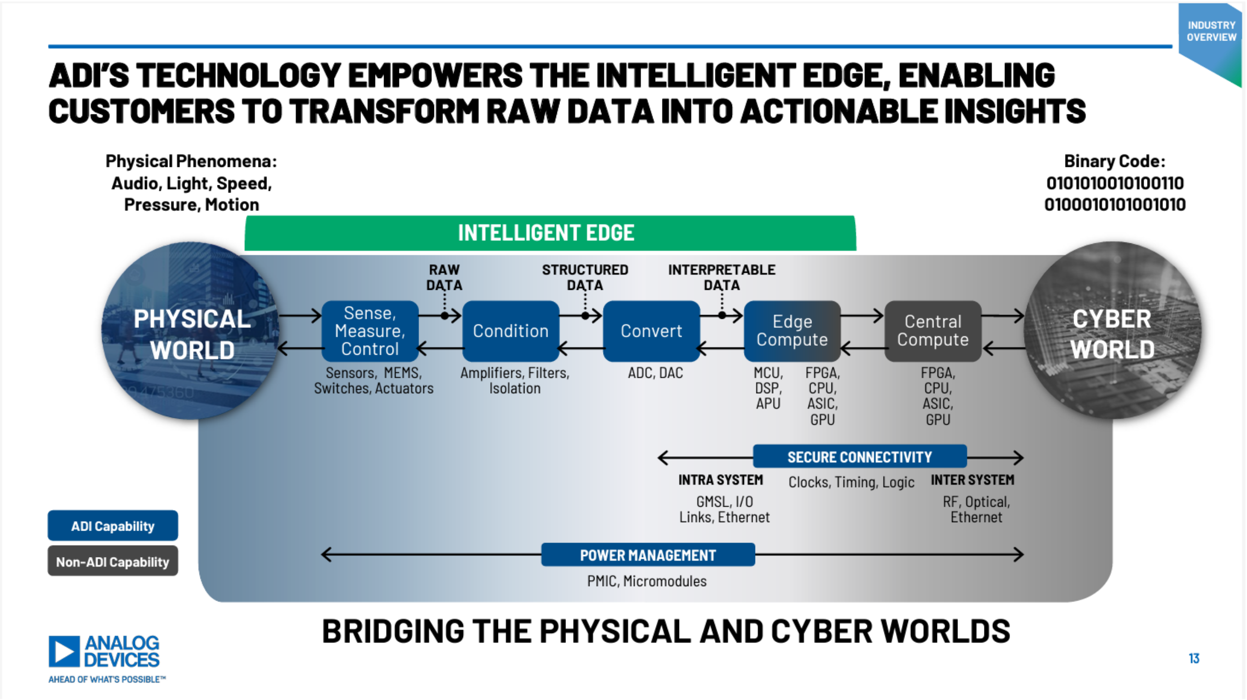
RNM Coding Guidelines for Emulation

Results

Summary

Analog Devices – Investing in Intelligent Edge Systems





Source: ADI Investor Relations → Company Overview → ADI Overview: The Bedrock of the Modern Digital Economy (https://investor.analog.com/static-files/8bd2c3d8-1401-45a2-868e-76fd82118f9f), FY 2023

Pushing the Mixed-Signal SoC boundary in Communications, DSPs, ADEF, Industrial

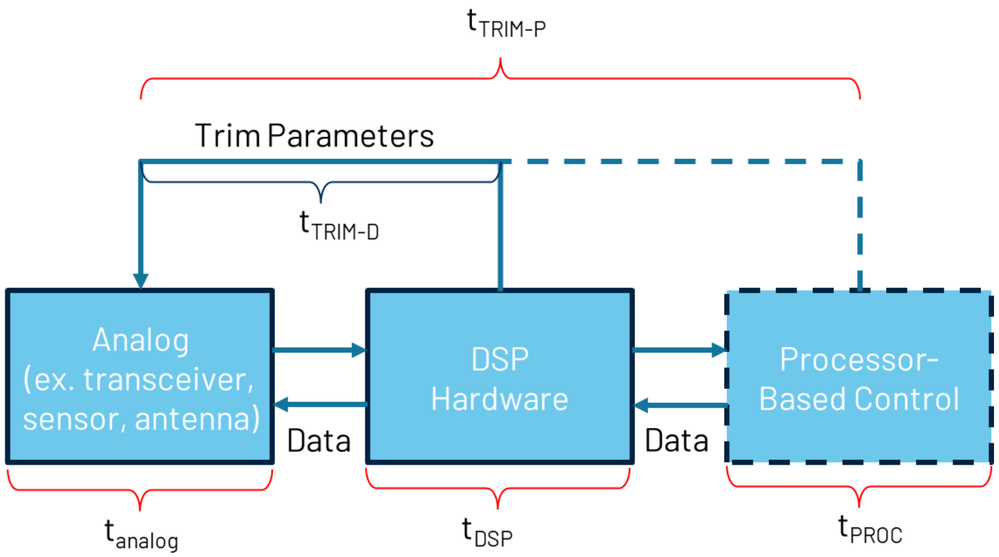
Rapidly increasing software and security content

R&D investment in enterprise systems for HW-based verification and validation since 2018

Verification of Digitally Controlled Mixed-Signal Systems

Long (wall clock) MS sims needed to verify trim/calibration

- Analog tuned by HW and/or HW/SW DSP



Key use case parameters

- t_{analog}: runtime of analog model
- t_{DSP}: runtime of DSP model
- t_{PROC}: runtime of SW process
- t_{TRIM-D}: duty-cycle for DSP HW trim
- t_{TRIM-P}: duty-cycle for HW/SW trim

Minimizing sim time requires understanding time spent in each parameter

- Assuming 1GHz system clock, if t_{TRIM-D} = 0.1s real-time, t_{DSP} sim runtime = 27 hrs, t_{analog} < 2.7 hrs
- Parameters differ per project, but the calculation above drives requirement for analog model runtime

Previous Approaches to Mixed-Signal Emulation

Black-box or Gray-box RNMs / Behavioral Models

- Cannot verify calibration loops or end-to-end configuration in emulation
- Gray-boxing makes some configuration verification possible but ad-hoc and very limited

Fixed-Point Behavioral Modeling

- Reasonable speed but lot more effort to maintain and validate extra set of models
- Large emulation footprint

Domain Specific Langauges (DSLs)

- Learning curve and extra effort to create new set of models
- Compilers/converters to target various end-platforms (simulation/emulation/FPGA)
- Suffers from same drawbacks as fixed-point for emulation systems

Technical Objectives

Enable Digital Mixed-Signal (DMS) emulation via synthesizable RNMs

Seamlessly retarget RNMs built for simulation to emulation platform

Enhance the Palladium compiler to support

- Real datatypes, User-Defined Nettypes (UDNs)
- # delays (procedural and continuous assignment)
- Datatype coercion
- Complex floating-point operations
- Provide coding guidelines to overcome emulation limitations
- Minimize emulation area overhead and throughput impact
- Prove approach on real-life, complex mixed-signal SoC

Demonstrate success using real-life mixed-signal system

Coding Guidelines and Considerations - I

Datatype support

- 'real' datatypes such as $real$, $shortreal$, $realtime$
- unpacked arrays and structs
- SV-2012 nettypes, including synthesizable resolution functions
- Fixed-arrays but not dynamic arrays because they aren't synthesizable

Delay support

- Pound (#) delays are typically not support but these are:
  - Rise/Fall/Off on Verilog gate primitives must also be transformed
- $not\ \#(\ RiseDelay, FallDelay\ )\ (\ O, o\_reg\ )$; transformed to:

- always process with time delay and no event.
  Example:

```
always #10 a = b;
```

- always process with time delay and explicit event.
  Example:

```
always @(a or b) begin
  #10;
  y = a + b;
end
```

- Continuous assignment with a single delay.
  Example:

```
assign #10 y = a + b;
```

```
always @(posedge o_reg) begin
  #RiseDelay;
  o_temp = ~o_reg
end
always @(negedge o_reg) begin
  #FallDelay;
  o_temp = ~o_reg;
end
assign O = o_temp;
```

Coding Guidelines and Considerations - II

Floating-point operations, math functions and conversion functions

- Operators with real operands in Table 11-1 in the IEEE Std 1800-2017 are supported in DUT except for the inside operator
- $\$rtoi, \$itor, \$realtobits, \$bitstoreal, \$shortrealtobits, \$bitstoshortreal$ are all supported

DMS Connectivity

- Simulators have features not in the SV LRM to help with DMS connectivity mismatches
- Emulation compiler has datatype coercion to a great degree, but not 1:1 with simulation
- VPI and a post-processing Python script created to trace every connection
  - Automation should be added to the emulation compiler

General limitations of synthesizable code apply including

- Ex: Verilog gate primitives, dynamic arrays, force and release operations of real datatypes, etc.

Results – Charge Pump PLL (CP-PLL) - I

Vehicle to test enhanced compiler and DMS support

Filter implemented as complex $H(s)$ transfer function with poles and zeros via bilinear transform

Feedback modeling, # delays in PFD block

wrealsum UDN for charge pump current modeling

Datatype coercion supported, enabling netlist reuse

Convert dynamic objects to static – coefficient arrays

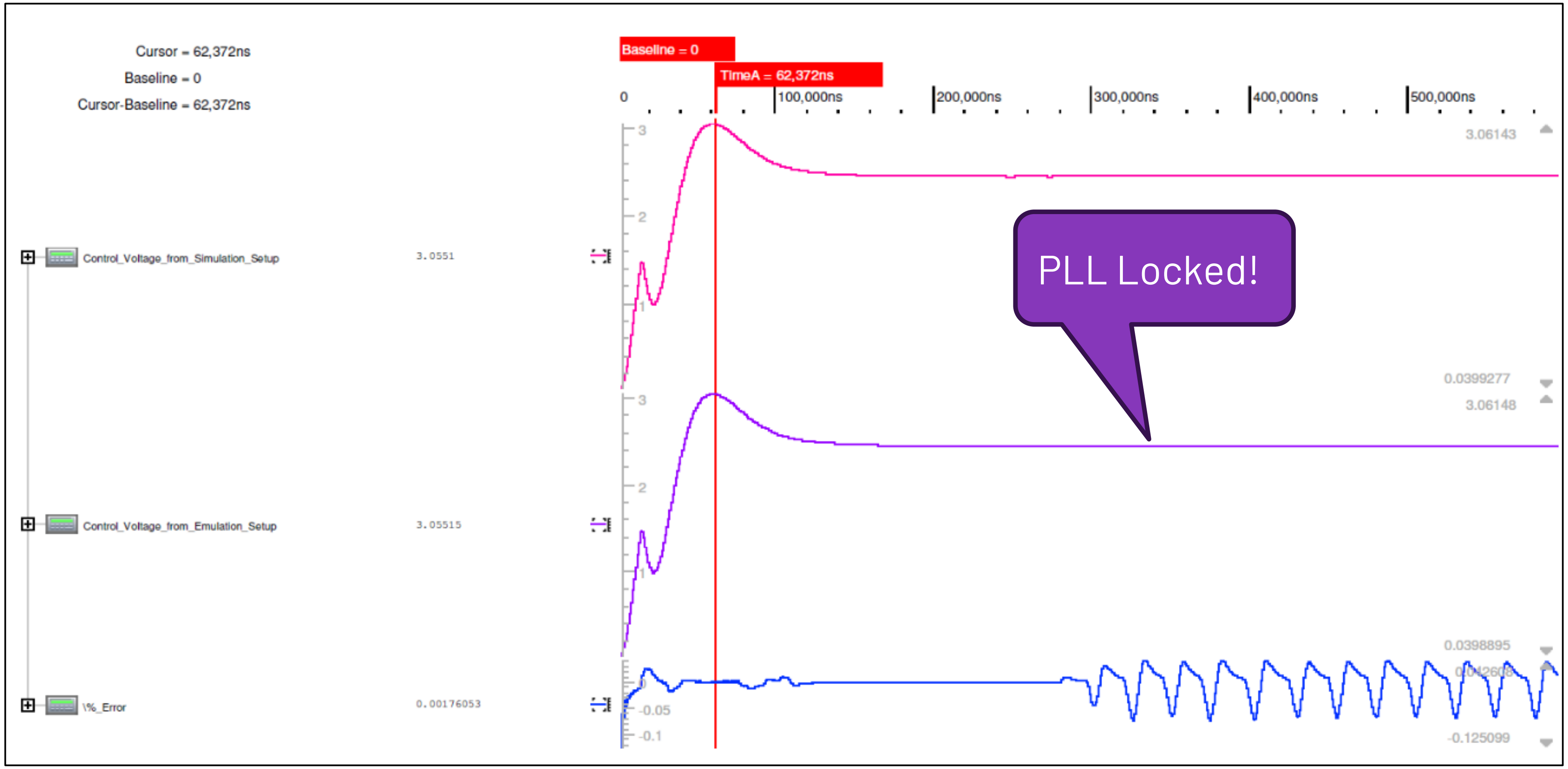
Required additional compiler arguments and options

```
// Library - pll, Cell - filter, View - schematic_behav

module filter (
output wrealsum  out,
input  wrealsum  in );

wrealsum i0_O;

ccvs  #( .a_sv('{1, 44e-6}), .b_sv('{0, 22.22, 146.08e-6, 193.6e-12}),
     .dim(1), .k("1e9"), .s2z_fs("1G"), .source_type("s-domain"), .AP(2), .BP(4))
   e0 (.NIN(nin), .NOUT(i0_O), .PIN(in), .POUT(out));
gn_vref  #( .voltage("0"), .conn(0))  i0 ( .O(i0_O));

endmodule
```

```
module pll (  );

wire vss, i4_out, vc;  /*don't need to be declared wrealsum as coercion is supported*/
bit ckin ;
wire logic ckdiv ;
wire logic ckout ;
wire up, i0_OB, down ;

pfd i1 ( .down(down), .up(up), .ckdiv(ckdiv), .ckin(ckin));
vco i2 ( .out(ckout), .vc(vc));
filter i3 ( .out(vc), .in(i4_out));
cpump i4 ( .out(i4_out), .down(down), .up(up));
divby32 i5 ( .ckout(ckdiv), .ckin(ckout));
lclock i0 ( .ckin(ckin), .i0_OB(i0_OB));
gn_vref  #( .voltage("0"), .conn(0))  i7 ( .O(vss));

`ifdef IXCOM_COMPILE
IXCclkgen #(500) ckg(mclk);  // 1ns CAKE master clock
`endif

endmodule
```

Results – Charge Pump PLL (CP-PLL) - II

CP-PLL Locks Successfully in Emulation!



Error < 0.04% vs. simulation

Results - 4T4R Transceiver - I

4T4R transceiver with application class ARM cores

Current emulation approach

- Gray-box or Black-box all AMS
- Simulation Acceleration mode with AVIPs

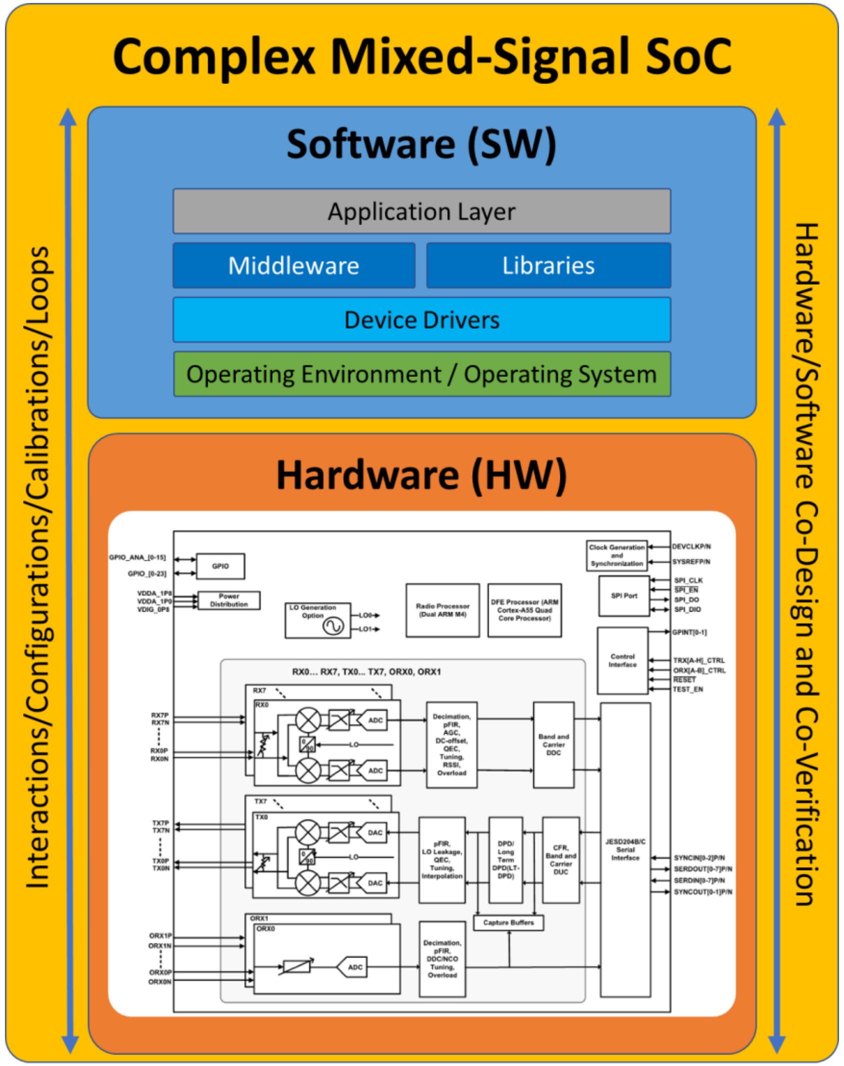
White-box bias generator block (hierarchical RNMs)

RNMs from ADI library, Extracted LUT, Hand-coded SV

Validated model on all platforms vs. transistor circuit

Integrated model into system-level emulation env

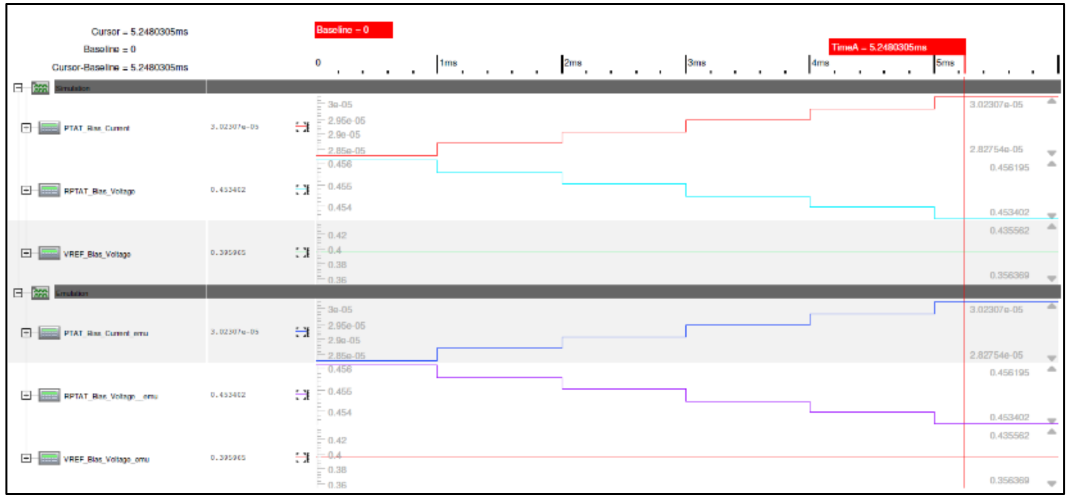
Self-trim initiated via register write by ARM core



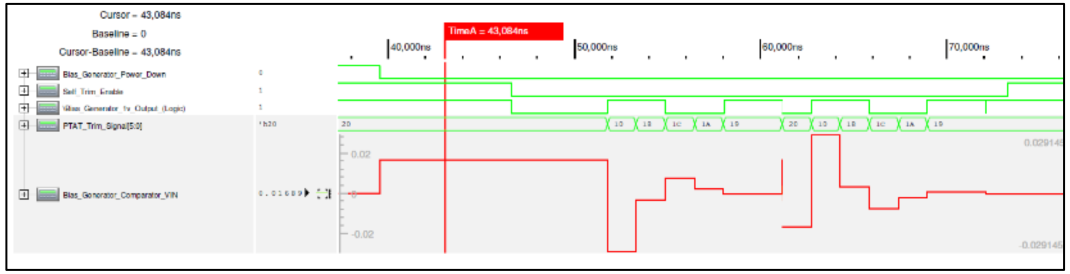
Results – 4T4R Transceiver - II

Simulation and Emulation Results Align!

Transceiver bias generator, Simulation vs. Emulation

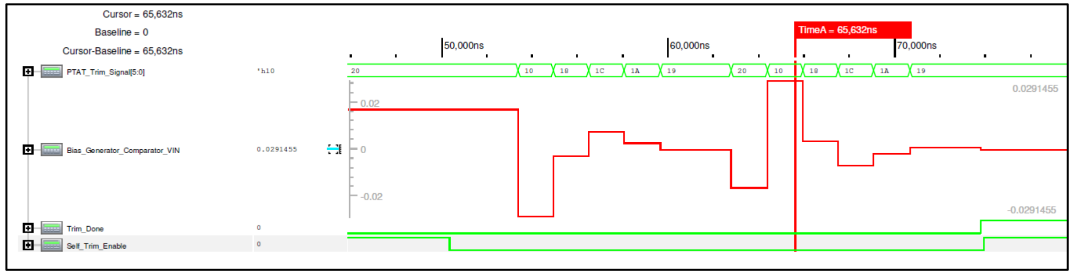


Simulation results for the bias generator self-trim routine



Emulation results for the bias generator self-trim routine

- 33% Throughput Impact
- Runtime – Simulation: 14400s   Emulation: 45s



320x speedup vs. simulation!

Summary

Innovations in compiler, emulation hardware, coding guidelines enable true DMS synthesis

Synthesize RNMs to floating-point fidelity with DMS features without recoding

CP-PLL effectively pipe-cleaned new solution, demonstrates synthesized RNM fidelity

Transceiver SoC tested out new approach in production on real-life SoC

320x speed-up on bias generator use-case with throughput impact well within tolerance

Enhanced ADI behavioral modeling library targeted at both simulation and emulation

Limitations: CM insertion, dynamic objects, Verilog gate primitives

No new methodologies or tools - all fits within current, well-established solutions

Key Takeaways

Synthesize RNMs Directly, no Black-Boxing or Recoding required!

Successfully Emulate Complex, Mixed-Signal HW+SW Systems!

Comprehensive Pre-Silicon HW Verification and SW Validation!

AHEAD OF WHAT'S POSSIBLE

analog.com

Q & A

2024
DESIGN AND VERIFICATION™
DVCON
CONFERENCE AND EXHIBITION
UNITED STATES
SAN JOSE, CA, USA
MARCH 4-7, 2024

cādence®