

Liberty

Liberty™ Delay Calculation Advancements

Bill Mullen
Group Director, R&D

Synopsys Interoperability Forum
October 21, 2004



> *Your Design Partner*

SYNOPSYS®

Delay Calculation

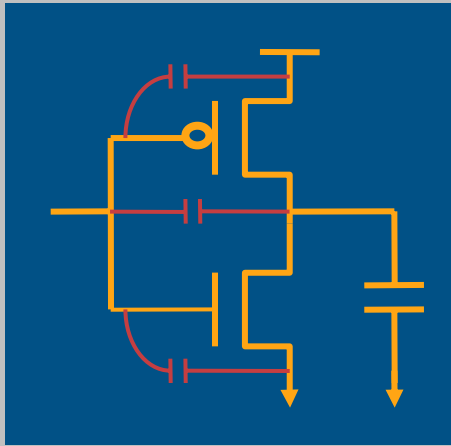
- Need to efficiently and accurately compute the response of a cell plus interconnect
 - Model the drive capability of the cell
 - Model the capacitance of the load pins
 - Replace parasitics with a Reduced-Order Model such as Block Arnoldi



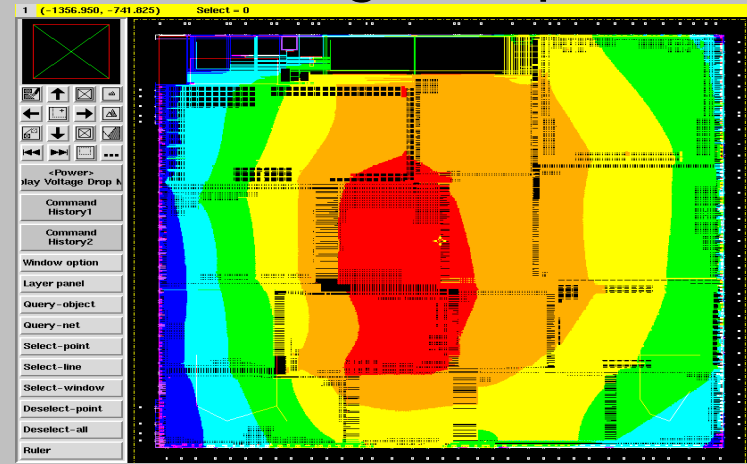
Determine signal waveforms at D1/Z, L1/A, L2/A, L3/A

Emerging Delay Calculation Needs

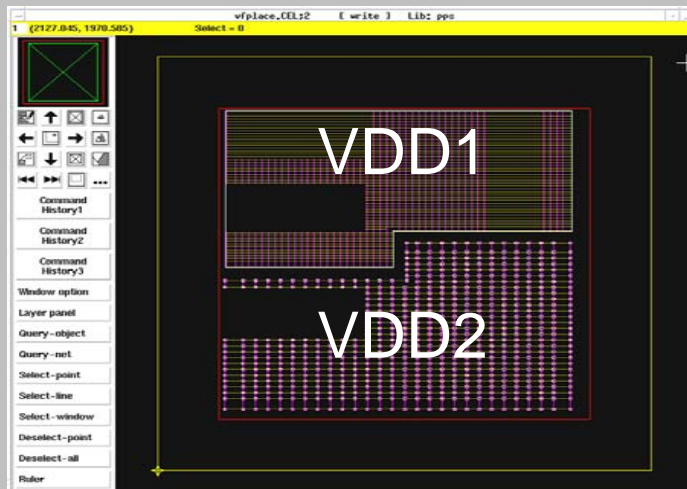
Complex Input Capacitance



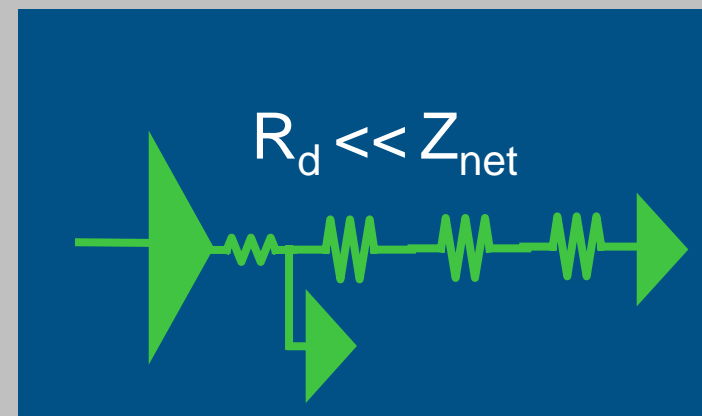
Voltage Drop



Voltage Islands

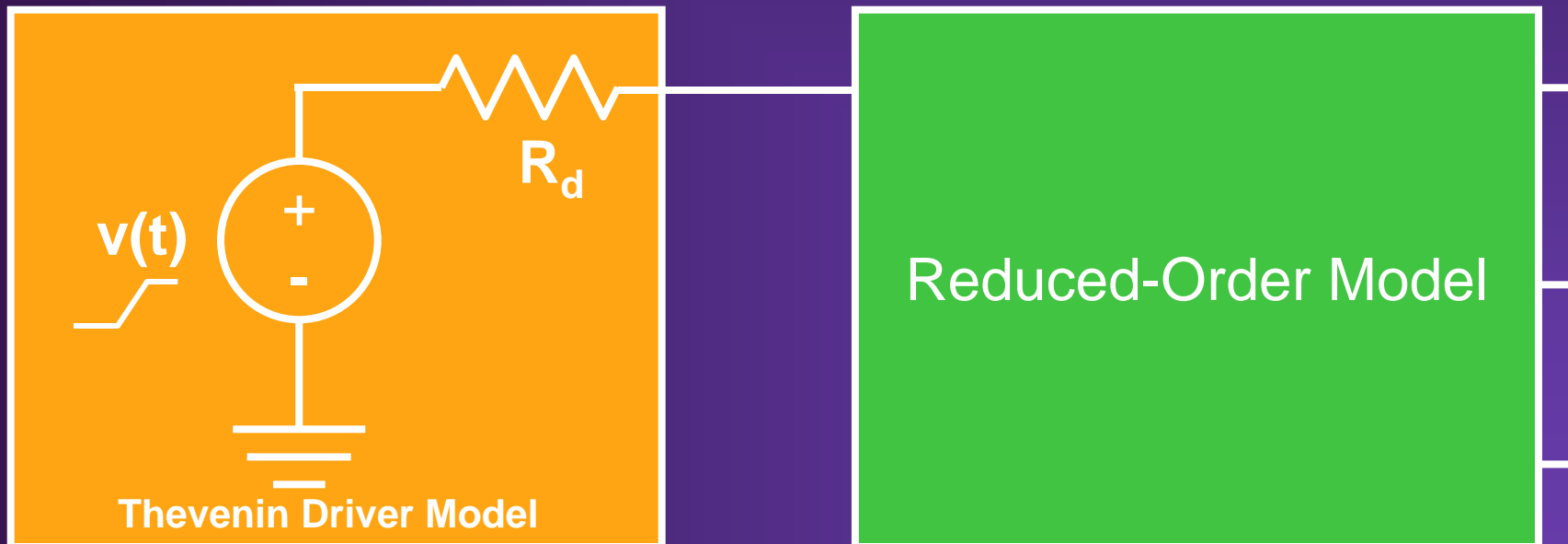


High Impedance Nets

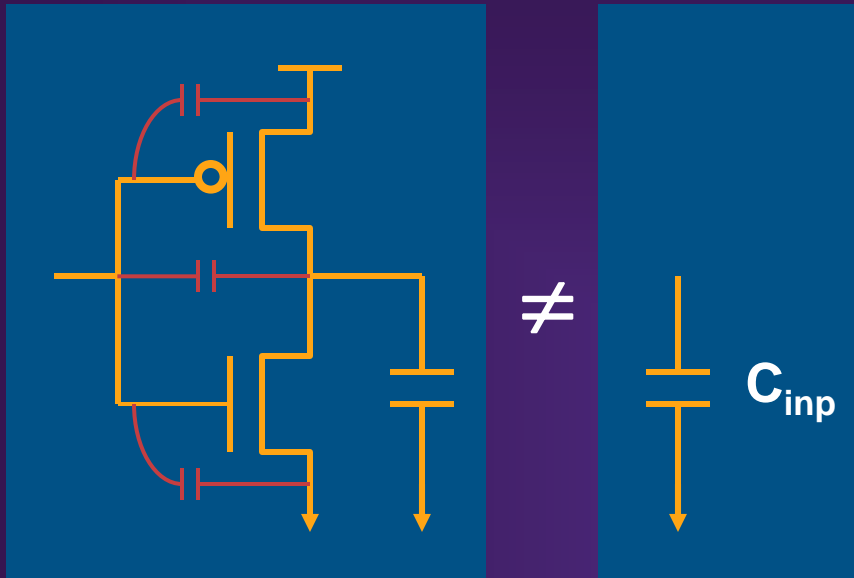


Previous Driver Model

- Ramp voltage source with fixed drive resistance
- Fast to calculate, and accurate for most nets
- Limited accuracy for large driver on high-impedance net



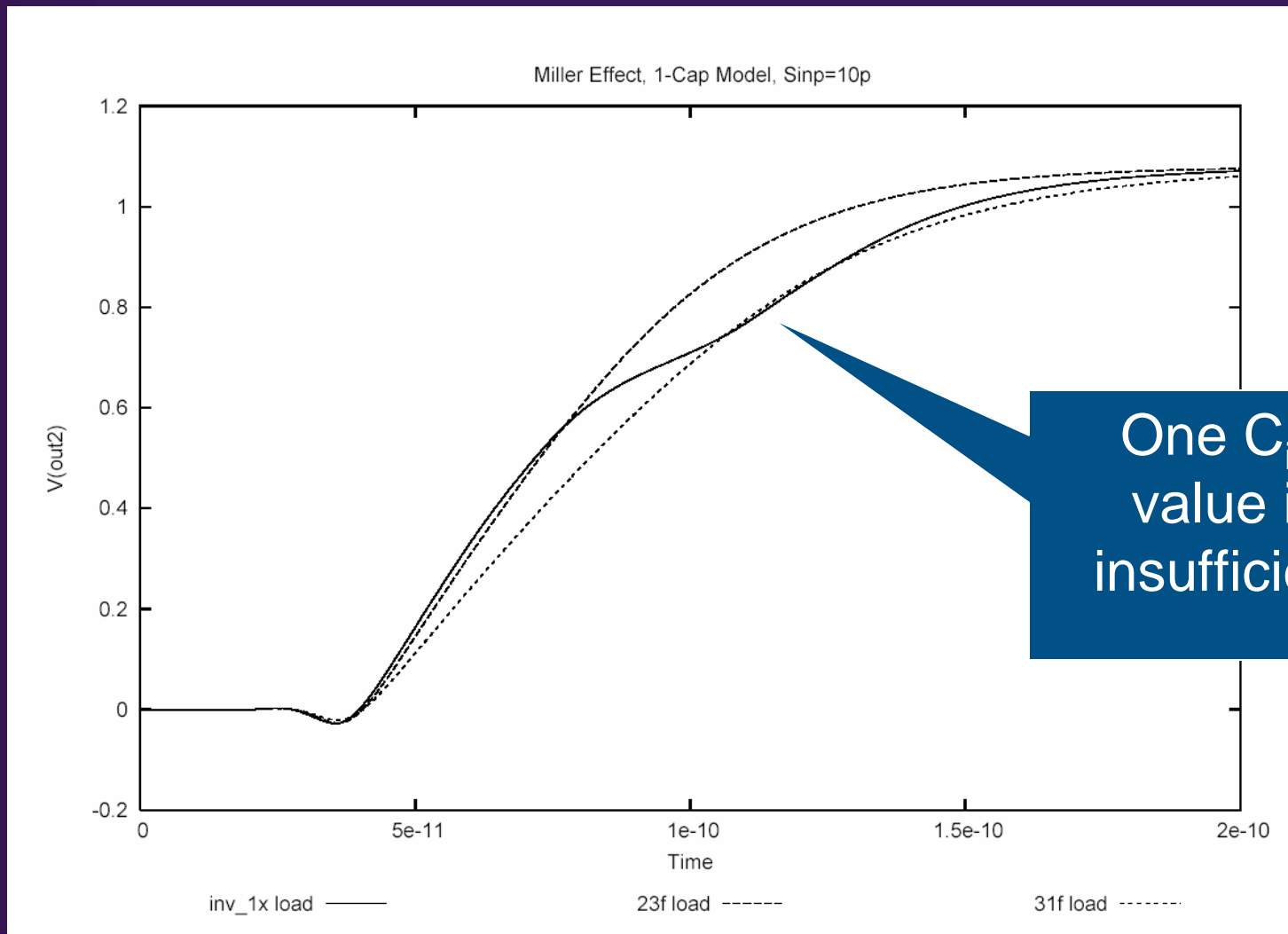
Receiver Modeling (Input pin cap)



- Previous approaches are insufficient:
 - Single cap or min/max rise/fall
- Capacitance varies during the transition
- Value depends on Voltage, Output load, Slew, State of the cell

Capacitance	Cut-off	Linear	Saturation
C_{gb}	$C_{ox} WL$	0	0
C_{gd}	$C_{ox} WL_D$	$\frac{1}{2} C_{ox} WL + C_{ox} WL_D$	$C_{ox} WL_D$
C_{gs}	$C_{ox} WL_D$	$\frac{1}{2} C_{ox} WL + C_{ox} WL_D$	$\frac{2}{3} C_{ox} WL + C_{ox} WL_D$

Previous Receiver Model: Single Capacitance Value



Liberty

CCS: Composite Current Source Delay Calculation



> Your Design Partner

SYNOPSYS®

CCS Driver Model

- **Composite Current Source** driver model is a time-varying, voltage-dependent nonlinear current source
- Works with any net topology, including high impedance nets
- Works with complex cells, including those with stacked transistors
- Works with non-monotonic behavior, including the Miller effect



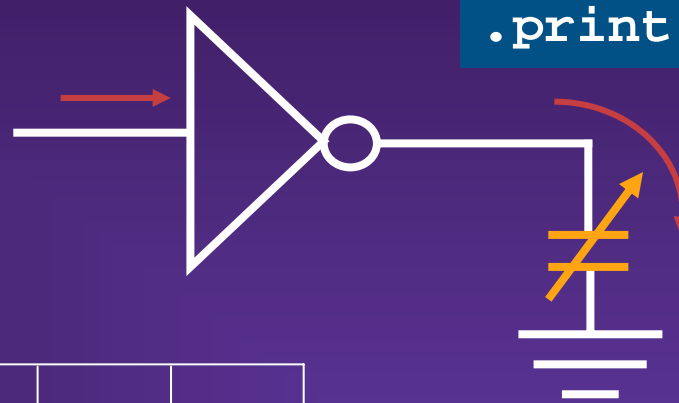
CCS Receiver Model

- **Represent C_{inp} as 2 values, with change at the delay threshold (such as 50% Vdd)**
 - $C1 = f(S_{inp}, C_{out})$
 - $C2 = f(S_{inp}, C_{out})$
 - Separate values for rise and fall
- **Specify on library arc or library pin**
 - Arc: Supports state dependence
 - Pin: When no forward timing arcs, such as D pin of flip-flop
- **Works with CCS driver model**
 - Dynamically adjust C_{inp} during the transition

Characterization for CCS

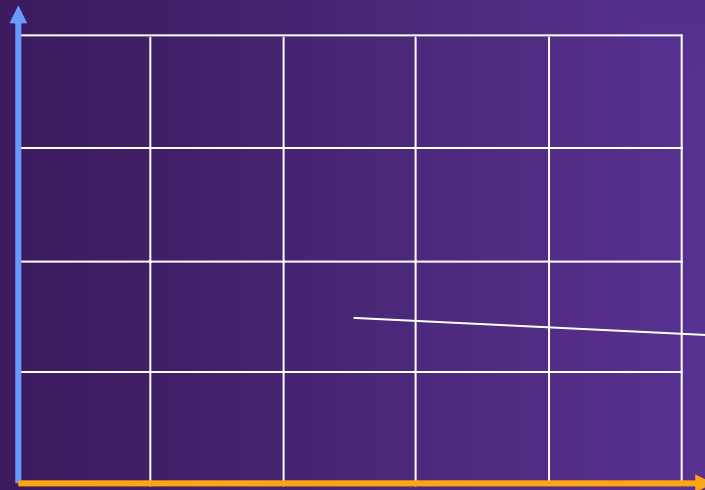
Same slew and capacitance indices as NLDM

```
.print i(Vinp)
```



```
.print i(Cout)
```

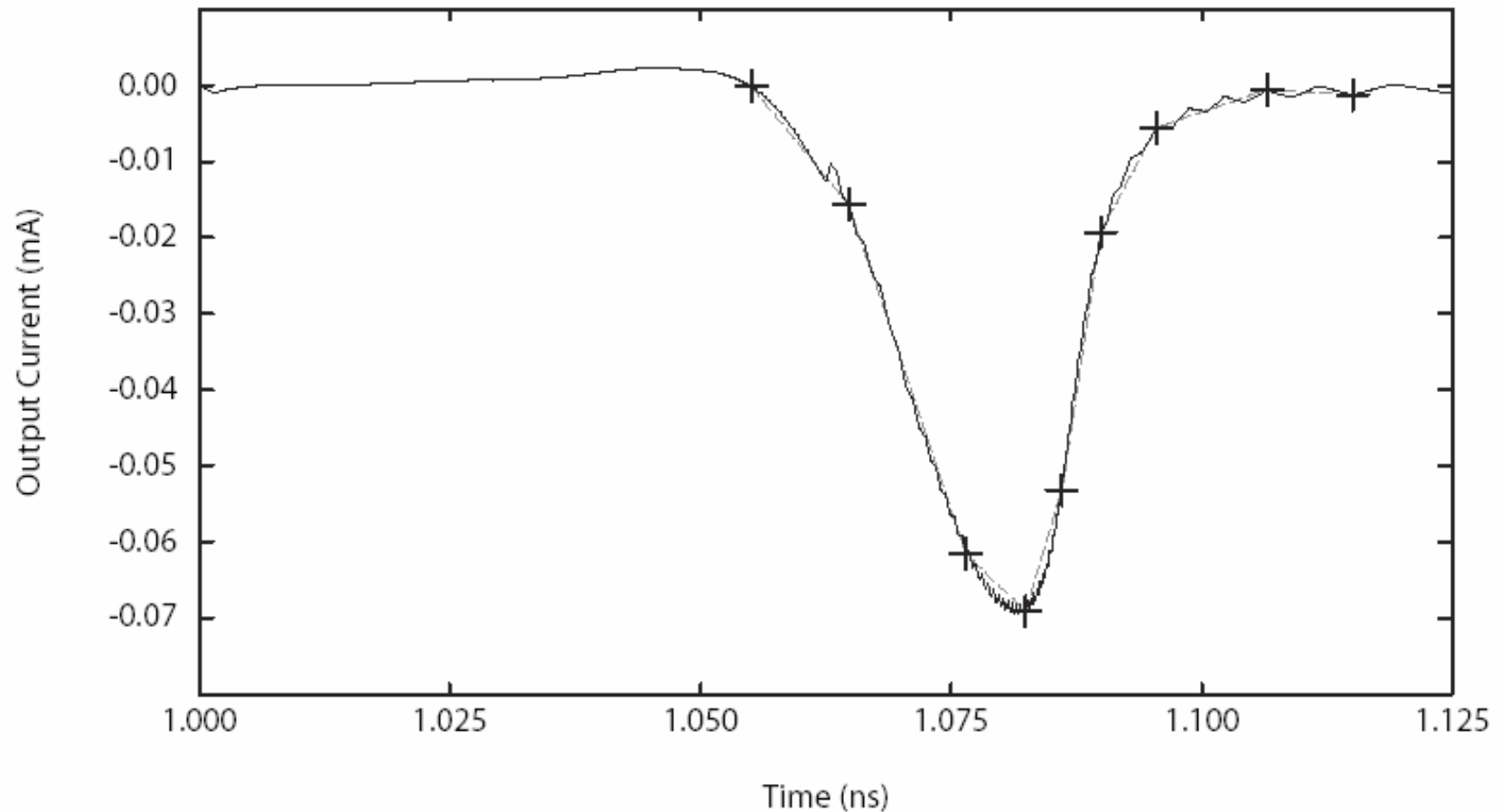
input
slew



output cap

- $i(t) : Cout$
- $i(t) : Vinp$

Example C_{out} Current Data



Liberty Syntax for Driver Model

```
output_current_template (CCST) {
    variable_1: input_net_transition;
    variable_2: total_output_net_capacitance;
    variable_3: time;
    index_1("0.1, 0.2");
    index_2("1.0, 2.0");
    index_3("1, 2, 3, 4, 5");
}
timing() {
    output_current_rise(CCST) {
        vector() {
            reference_time : 0.05;    # input waveform threshold crossing
            index_1(0.1);              # Sinp
            index_2(2);                # Cout
            index_3(1.1, 1.2, 1.3, 1.4, 1.5); # time
            values(0.1, 0.2, 0.3, 0.1, 0.05); # current values
        }
        ...
    }
}
```

Liberty Syntax for Receiver Model

- **Pin-based: $f(S_{inp})$**

```
lu_table_template(LTT1) {
    variable_1: input_net_transition;
    index_1("0.1, 0.2, 0.3, 0.4");
}

pin(A) {
    direction : input;
    receiver_capacitance() {
        receiver_capacitance1_rise(LTT1) {
            values("1, 2, 3, 4");
        }
        receiver_capacitance1_fall(LTT1) {
            values("1, 2, 3, 4");
        }
        receiver_capacitance2_rise(LTT1) {
            values("1, 2, 3, 4");
        }
        receiver_capacitance2_fall(LTT1) {
            values("1, 2, 3, 4");
        }
    }
}
```

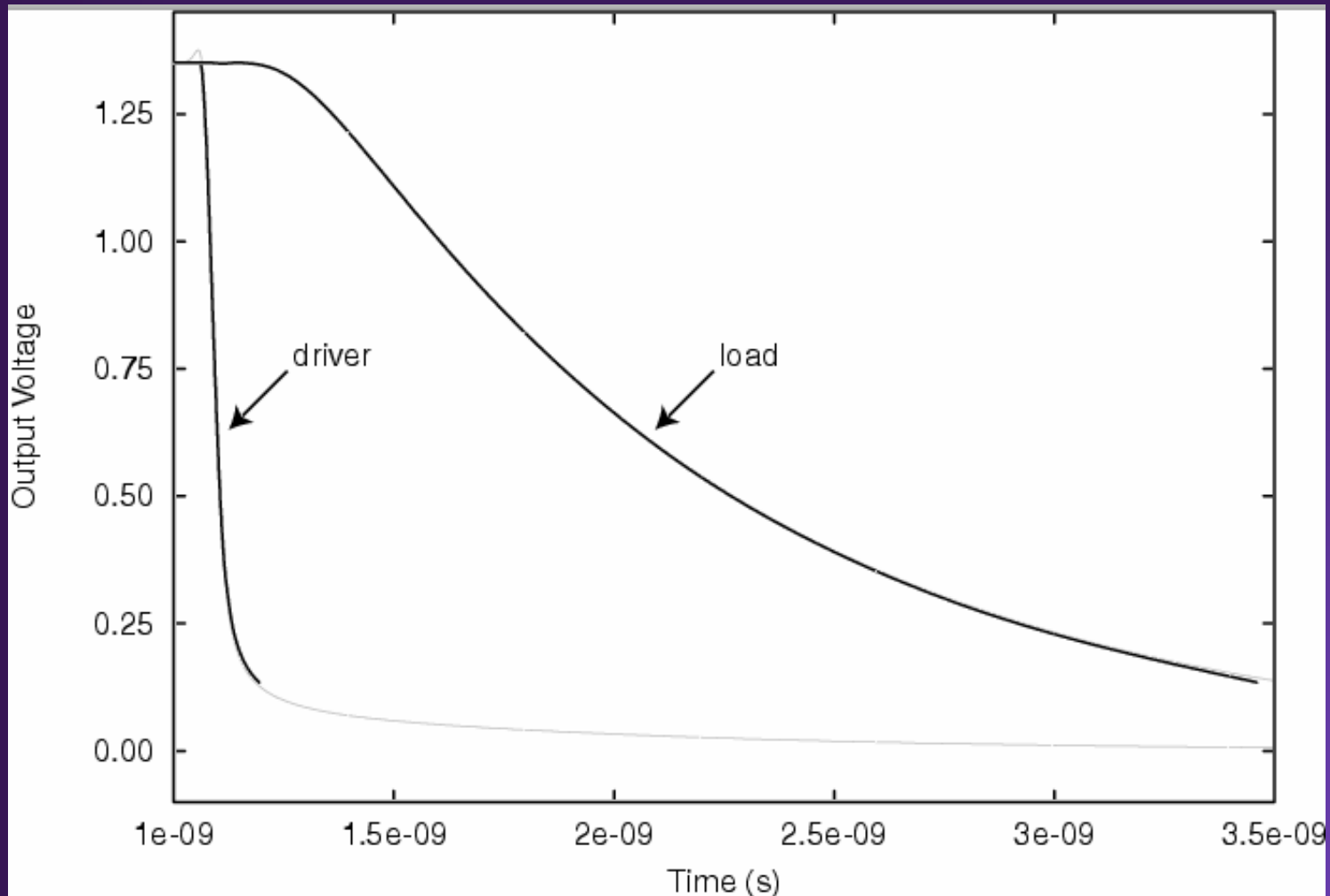
- **Arc-Based: $f(S_{inp}, C_{out})$**

```
lu_table_template(LTT2) {
    variable_1: input_net_transition;
    variable_2: total_output_net_capacitance;
    index_1("0.1, 0.2");
    index_2("1, 2"); }

timing() {
    related_pin : "B";
    receiver_capacitance1_rise(LTT2) {
        values("1, 2, 3, 4");
    }
    receiver_capacitance1_fall(LTT2) {
        values("1, 2, 3, 4");
    }
    receiver_capacitance2_rise(LTT2) {
        values("1, 2, 3, 4");
    }
    receiver_capacitance2_fall(LTT2) {
        values("1, 2, 3, 4");
    }
}
```

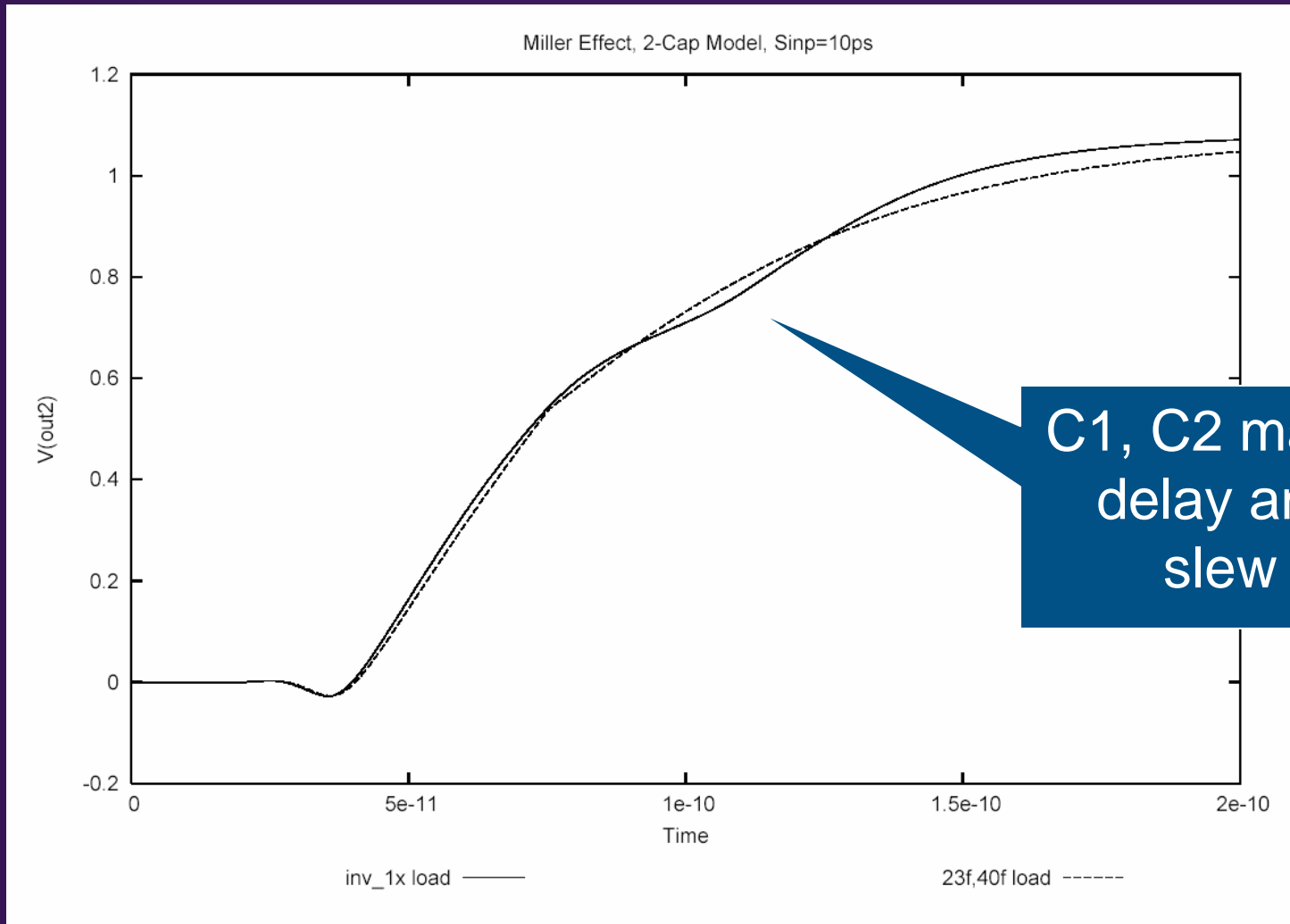
CCS Response vs. HSPICE®

Large driver, high-impedance net



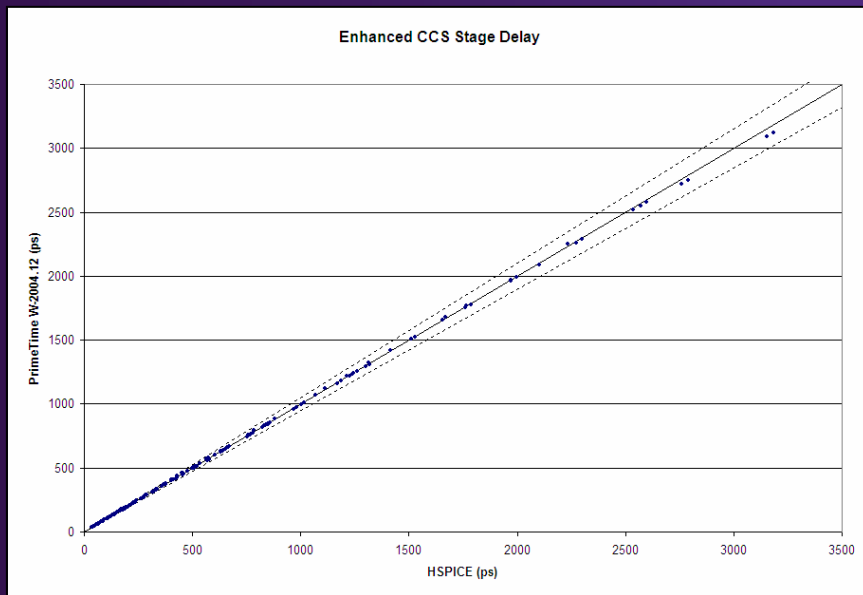
< 1ps error in delay

CCS Receiver Model Results

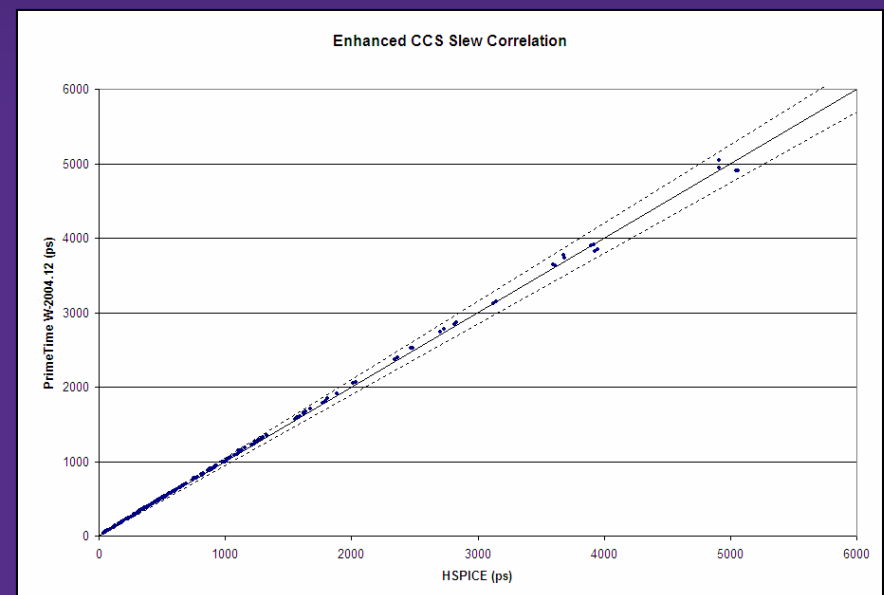


CCS: 2% versus HSPICE

- Results on testcases with varied driver size, wire length and input transition:



Delay

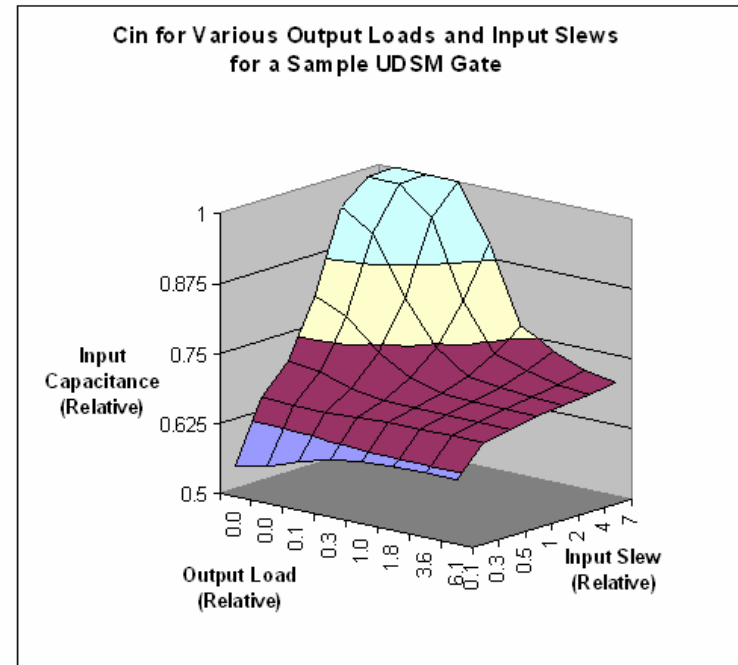


Slew



Benefits of CCS Models

- Reduced Design Margins
 - Improved spice correlation allows design margin reduction.
 - Pessimistic characterization data required to cover corner case conditions is reduced.
 - => Better design area, power, and design time.
- Improved Cycle Time
 - Reduces need to spice critical paths, clock trees, etc. required today on many designs.
- Low Characterization Overhead
 - Data extracted directly from existing characterization runs.
 - Low data volume overhead. Timing model libraries remain compact.



TI Technology Evaluation Only. No Mass-Deployment Commitment

Liberty Delay Calculation Summary

- **Liberty evolution for high accuracy delay calculation**
 - New CCS driver and receiver modeling support
 - Results within 2% of SPICE
 - Easy characterization – same SPICE runs as for NLDM
- **CCS available in Liberty 2004.12**
- **Tool Support of Liberty CCS**
 - Library Compiler™: 2004.12
 - PrimeTime®: 2004.12
 - Star-MTB™ (library characterization): Beta 2005.03
 - Physical Compiler® & Astro™: 2005