# New Way to Improve Power Supply Induced Jitter Simulation Accuracy for IBIS Model

Yifan Ding*, Yin Sun#, Zhiping Yang^, Chulsoon Hwang*

*Missouri S&T EMC Laboratory, #Zhejiang Lab, ^Waymo

IBIS Summit at
2021 IEEE Virtual Symposium on EMC+SIPI
August 12, 2021

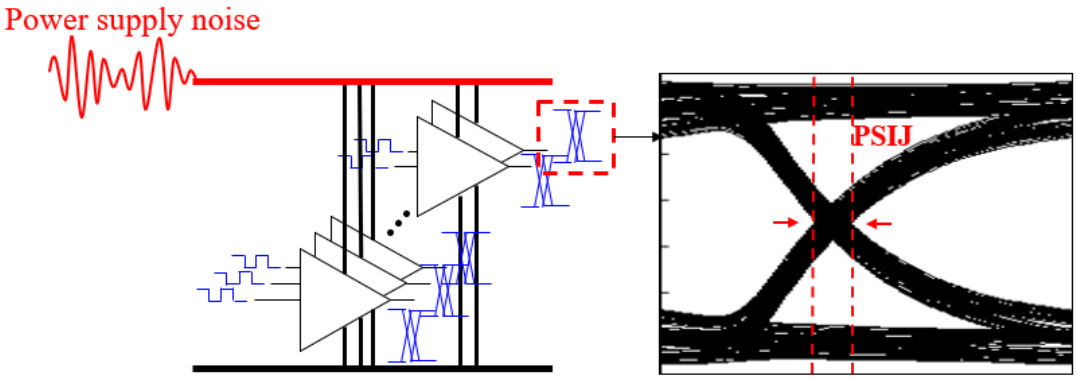
# Outline

- Introduction of  Power Supply Induced Jitter (PSIJ)
- Limitations of the Current Power-Aware IBIS Model
- Previous Proposed Behavior Model
- Feedbacks from IBIS ATM Group
- New Improvements
  - Accuracy Improvement for Ku/Kd Coefficient Extraction
  - Jitter Sensitivity Based Modification
  - Applied to Over-clocking Case
- Conclusions

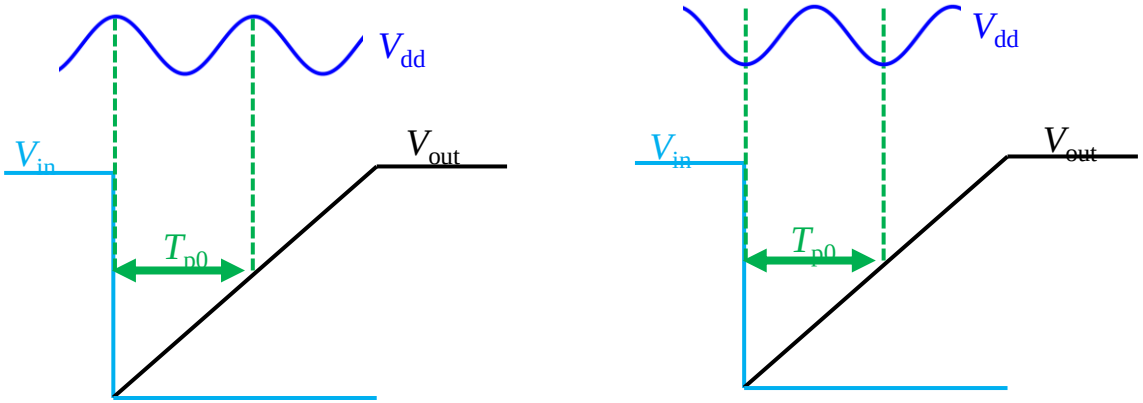
# Power Supply Induced Jitter (PSIJ)

Power supply induced jitter (PSIJ):

- The time variation in the output transition edges from ideal positions due to the voltage fluctuations on power rail.



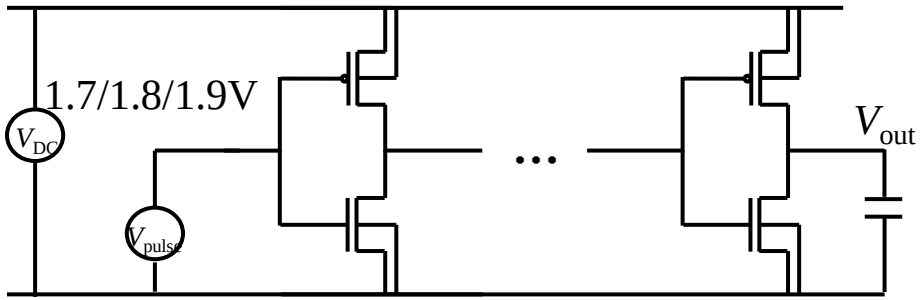

- The Vcc noise can take effect during the propagation delay time range;
- The influence is accumulated, just consider instantaneous voltage value is not accurate.



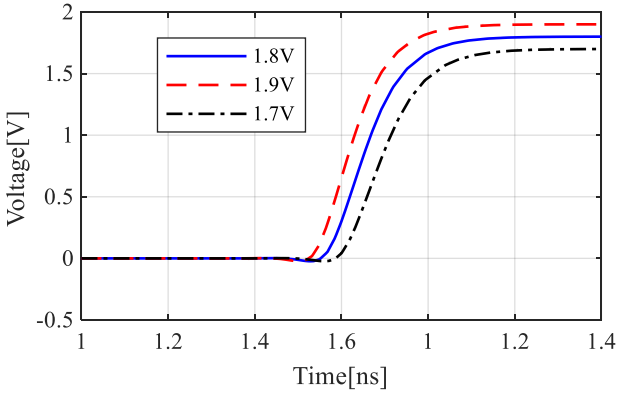
# Limitations of the Current Power-Aware IBIS Model

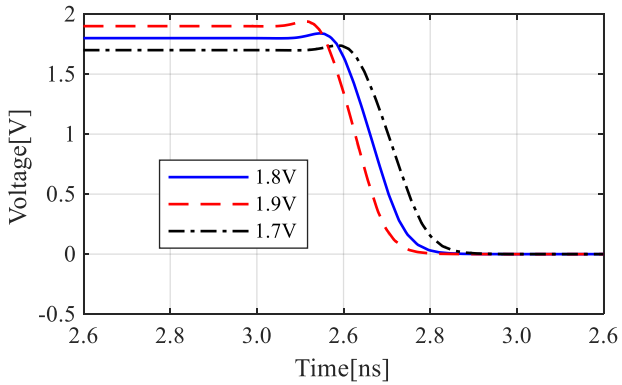
- **Cannot** account for the delay change caused by power noise correctly.

➢ Example: an inverter chain output, change power voltage to 1.7/1.8/1.9V, respectively

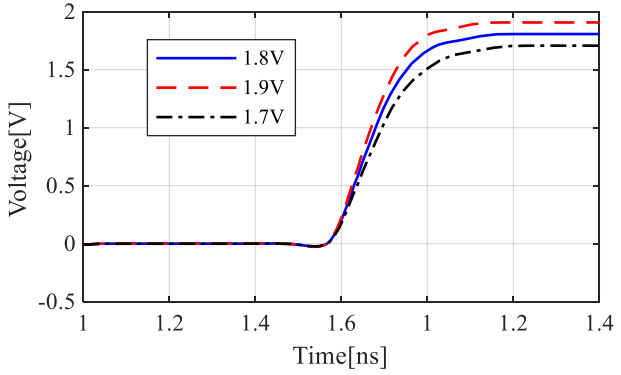


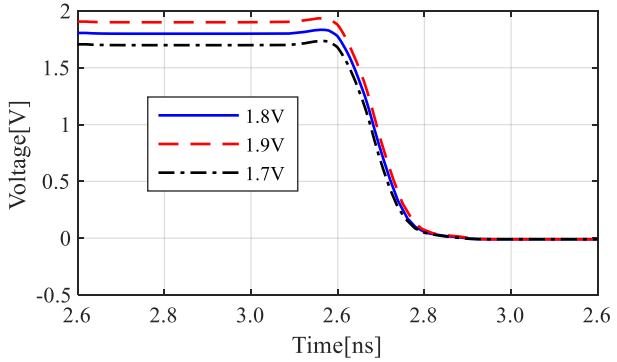

Spice Results




Power-aware IBIS model Results
(ver5.1,generated with EDA tool)



# Limitations of the Current Power-Aware IBIS Model

- Power-aware IBIS model considers gate modulation effect, ratio modification on Ku, Kd based on power rail voltage value

**Gate Modulation Coefficients**

The ST "Gate Modulation" solution is based on the introduction of two coefficients, one for the Pullup and one for the Pulldown stage, which modulate properly the IBIS standard current (I_IBIS-STD) when a bouncing noise occurs on the power and ground nodes

I(Vgs, Vds) = Kssn(Vgs,Vds)*I(Vgs=VDD,Vds)

Effective SPICE current — IBIS standard current

I_effective = Kssn(Vgs,Vds)*I_IBIS-STD

$$K_d(t)I_{pd} -> K_{sspd}(V_{pd})K_d(t)I_{pd}$$

$$K_u(t)I_{pu} -> K_{sspu}(V_{pu})K_u(t)I_{pu}$$

$$K_{sspd}(V_{pd}) = \frac{V_{pd}}{I_{sspd}(0)}$$

$$K_{sspu}(V_{pu}) = \frac{V_{pu}}{I_{sspu}(0)}$$

Source: "BIRD 98 and ST 'Gate Modulation' Convergence", IBIS Open Forum Teleconference, Jan. 26th, 2007
http://www.ibis.org/docs/BIRD98&ST_Proposal_Convergence.ppt

- The ratio modification Ksspd, Ksspu on Ku, Kd is only a function of $V_{pd}$ (Vcc-Vout) or $V_{pu}$ (Vout-Vgnd), it cannot reflect the effect of power rail voltage noise on switching edge timing change

Previous method on modification of Ku, Kd does not consider the time averaged effect;
Source: Behavioral modeling of jitter due to power supply noise for input/output buffers (US Patent 9842177B1)

# Previous Proposed Behavior Model

- Modify Ku(t), Kd(t) as a function of **<u>time averaged</u>** power rail voltage Vcc(t); introduce correction coefficient B and A as a function of **<u>time</u>**

$$K_u(t) = K_{u0}(t) + B_u(t) \cdot \left[\frac{\int_0^t V_{cc}(\tau)d\tau}{t} - V_{cc0}\right] + A_u(t)\left[\frac{\int_0^t V_{cc}(\tau)d\tau}{t} - V_{cc0}\right]^2$$

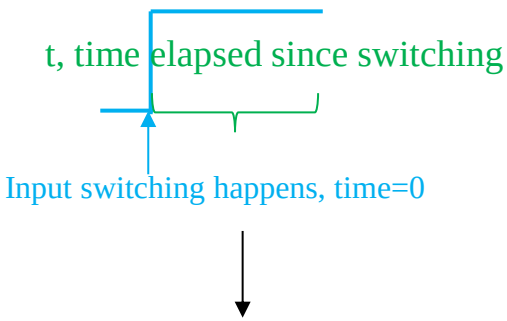
$$K_d(t) = K_{d0}(t) + B_d(t) \cdot \left[\frac{\int_0^t V_{cc}(\tau)d\tau}{t} - V_{cc0}\right] + A_d(t)\left[\frac{\int_0^t V_{cc}(\tau)d\tau}{t} - V_{cc0}\right]^2$$

Ku, Kd under nominal Vcc

Linear fitting coefficient

Quadratic fitting coefficient

Averaged Vcc(t) after the switching event happens;




Achieved by adding delay elements that store

- The time of switching edges
- Time averaged Vcc since switching event happens

- 2 equations, 2 unknowns' algorithm to extract Ku(t), Kd(t)

$$K_u(t)*I_u(V_1) + K_d(t)*I_d(V_1) = I_{out}(V_1)$$

$$K_u(t)*I_u(V_2) + K_d(t)*I_d(V_2) = I_{out}(V_2)$$

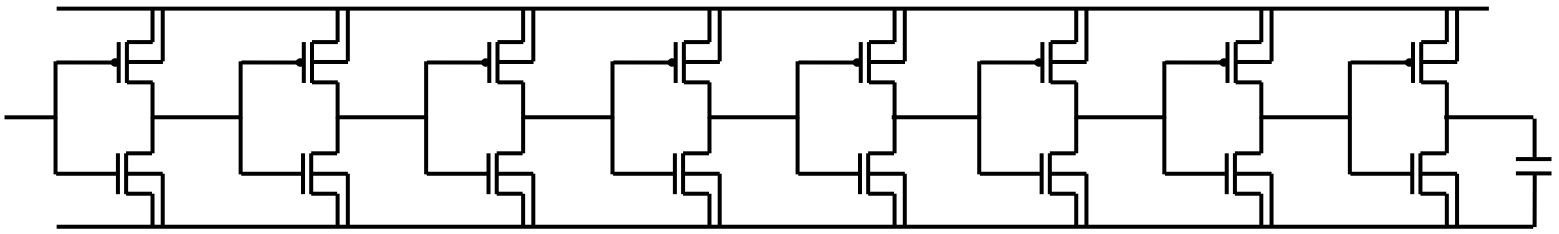
- 2 equations, 2 unknowns' algorithm to extract Bu(t), Au(t) and Bd(t), Ad(t)

$$K_{u\_max}(t) = K_{u0}(t) + B_u(t)(V_{cc\_max} - V_{cc0}) + A_u(t)(V_{cc\_max} - V_{cc0})^2$$

$$K_{u\_min}(t) = K_{u0}(t) + B_u(t)(V_{cc\_min} - V_{cc0}) + A_u(t)(V_{cc\_min} - V_{cc0})^2$$

# Previous Proposed Model Validation

- Tested driver

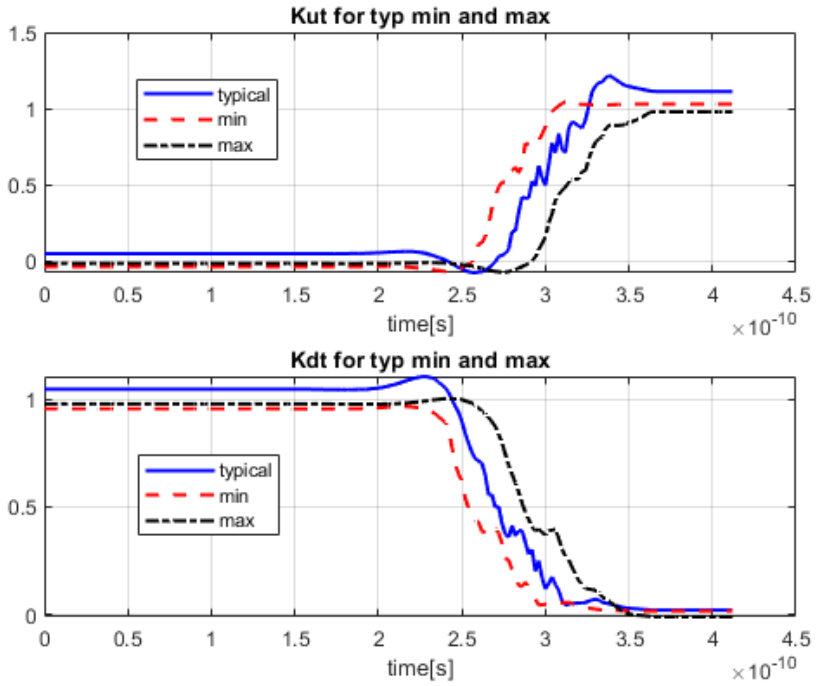


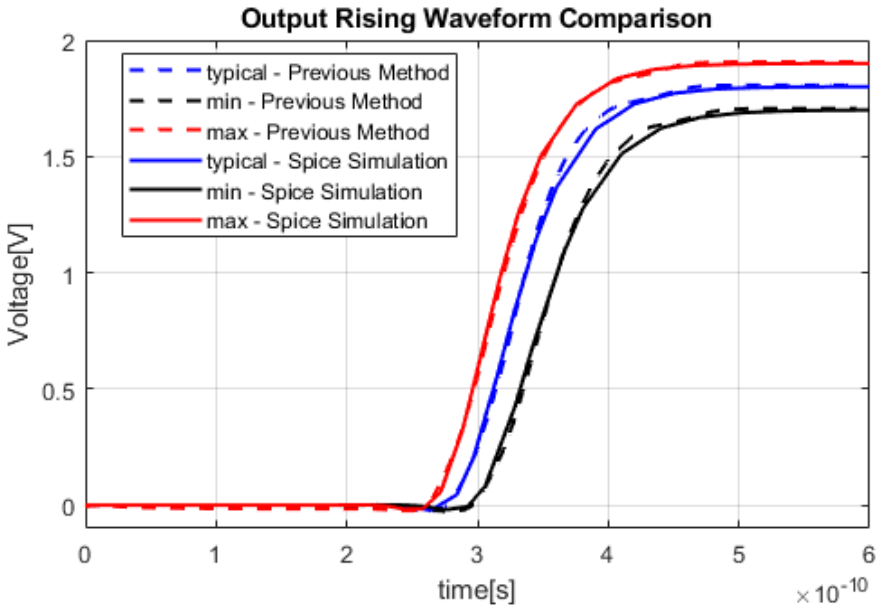

**M=1 M=2 M=4 M=8 M=16 M=32 M=64 M=128**

180nm technology, nominal voltage 1.8V

**< All NMOS>**
**nch_tn**
**W = 1 um**
**L = 180 nm**

**< All PMOS>**
**pch_tn**
**W = 2 um**
**L = 180 nm**



# Feedbacks from IBIS ATM Group

- Extracted initial or steady state value of Ku/Kd is not exactly 0 or 1 for the previous algorithm.

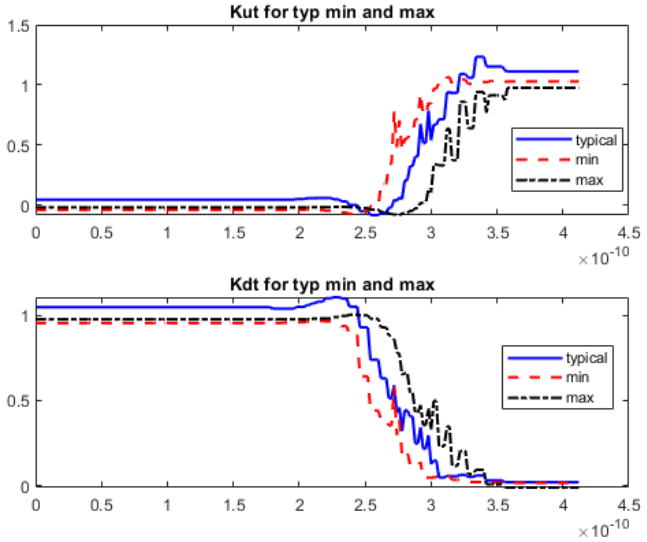
- Ku/Kd correction coefficients B and A are related to Process, Voltage and Temperature instead of only the supply voltage fluctuation.

- Algorithm is only for the case that driver propagation delay is smaller than the input switching period.

# Accuracy Improvement for Ku/Kd Coefficient Extraction

## Feedback 1

- Extracted initial or steady state value of Ku/Kd is not exactly 0 or 1 for the previous algorithm.




## Solution

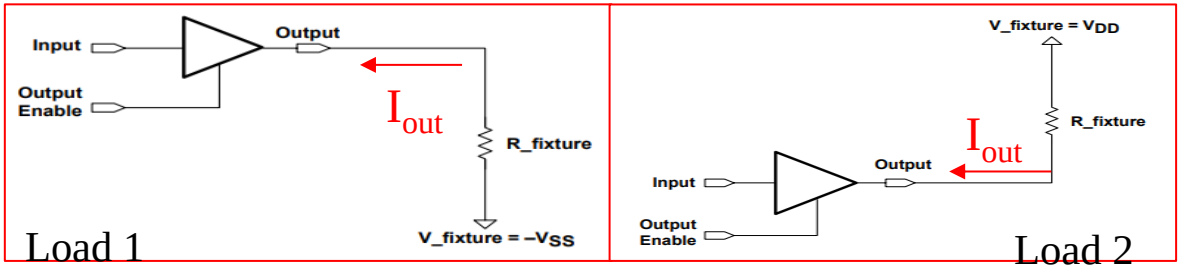
- Check parameters in IBIS model that are related to the Ku/Kd extraction.
- Use more accurate IBIS model.
  - 2 equations, 2 unknowns' algorithm to extract Ku(t), Kd(t)

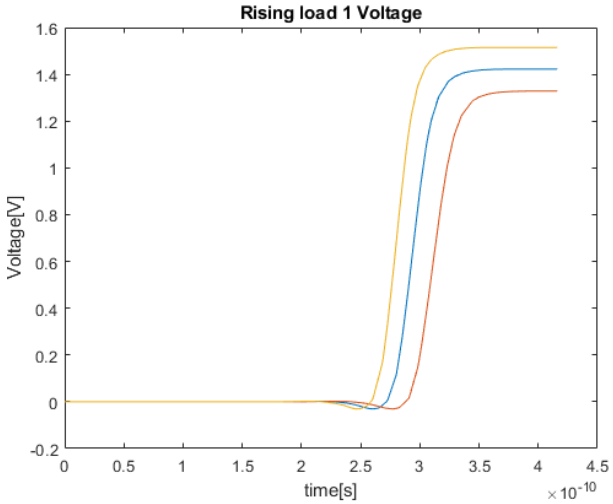
$$K_u(t)*I_u(V_1)+ K_d(t)*I_d(V_1)=I_{out}(V_1)$$
$$K_u(t)*I_u(V_2)+ K_d(t)*I_d(V_2)=I_{out}(V_2)$$

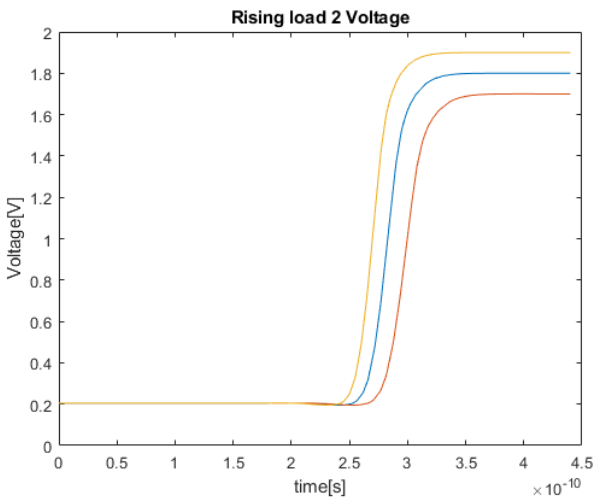
⇨ Check Iu, Id, and Iout

# Accuracy Improvement for Ku/Kd Coefficient Extraction

- Check Ku/Kd extraction Process




$$K_u(t)*I_u(V_1)+ K_d(t)*I_d(V_1)=I_{out}(V_1)$$
$$K_u(t)*I_u(V_2)+ K_d(t)*I_d(V_2)=I_{out}(V_2)$$

- IBIS I-V Table references are GND (Vcc) for models that have 0 current when Vout = GND (Vcc)

For rising edge
- Initial state, Vout1= GND, Ku(t) = 0, Kd(t) = 1, Ipd(V1) should be 0.
- Steady state, Vout2 = Vcc, Ku(t) = 1, Kd(t) = 0, Ipu(V2) should be 0.

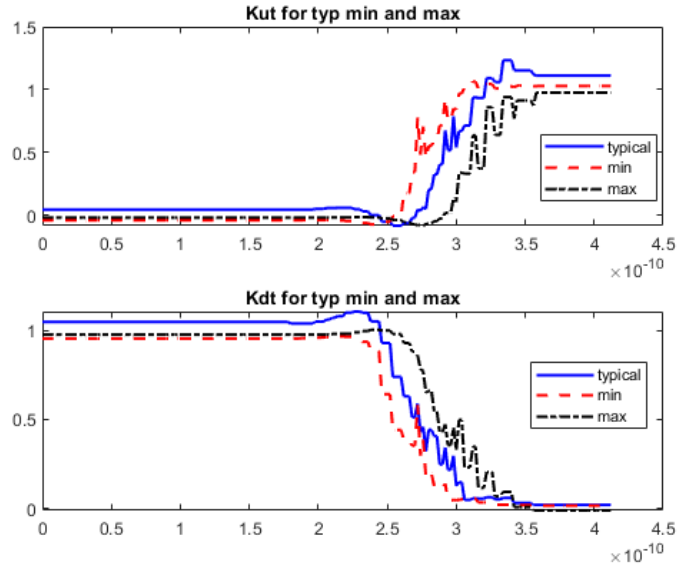
For falling edge,
- Initial state, Vout2= Vcc, Ku(t) = 0, Kd(t) = 1, Ipu(V2) should be 0.
- Steady state, Vout1= GND, Ku(t) = 1, Kd(t) = 0, Ipd(V1) should be 0.

# Accuracy Improvement for Ku/Kd Coefficient Extraction

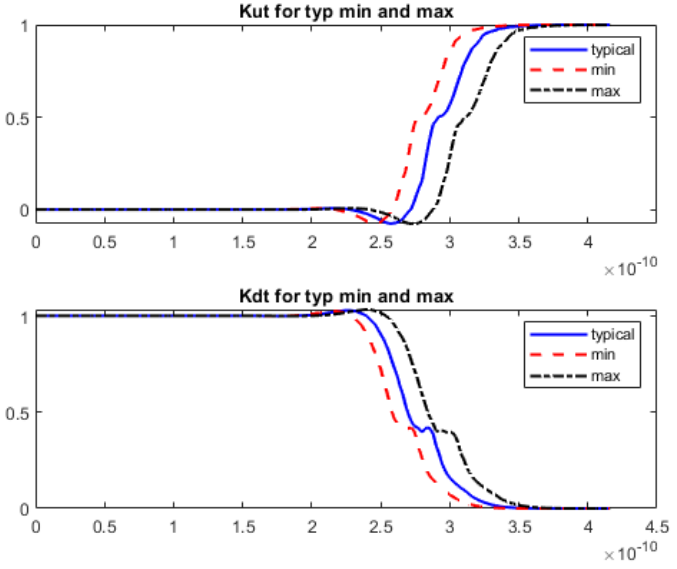
- Compare IBIS model extracted from two different tools

| Voltage at Reference | Simulation Tool 1 | Simulation Tool 2 |
|---|---|---|
| PU IV typ | -1.4mA@refV | 17.47uA@refV |
| PU IV min | 0.7mA@refV | 19.04uA@refV |
| PU IV max | 1.7mA@refV | 15.95uA@refV |
| PD IV typ | 2.5mA@refV | -8.29uA@refV |
| PD IV min | 1mA@refV | -12.15uA@refV |
| PD IV max | -3mA@refV | -4.92uA@refV |

- Rising edge Ku/Kd extracted from Tool 1



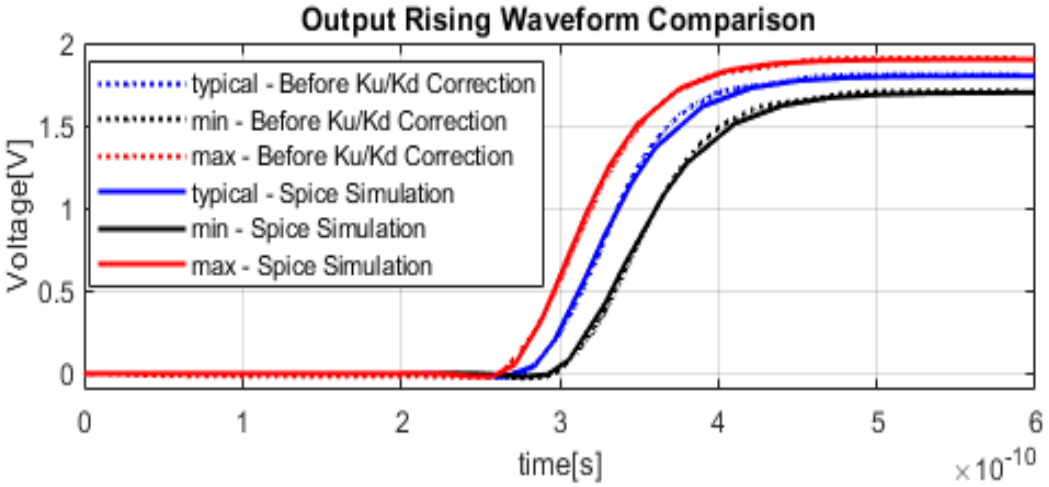

- Rising edge Ku/Kd extracted from Tool 2

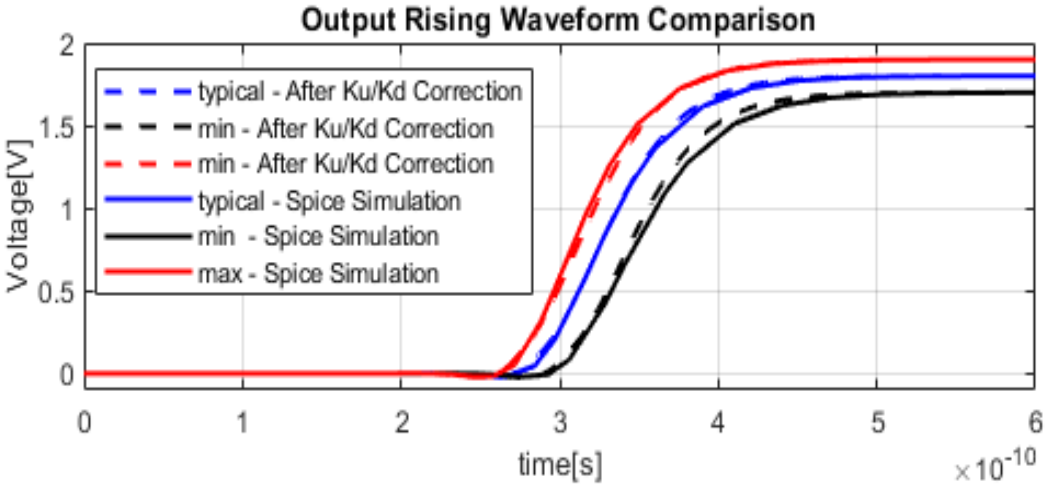


# Accuracy Improvement for Ku/Kd Coefficient Extraction

Output comparison for modification Before/After Ku/Kd correction

- Small offset of initial and steady state output value caused by Ku/Kd offset has been fixed.



# Jitter Sensitivity Based Modification

## Feedback 2

- Ku/Kd correction coefficients B and A are related to Process, Voltage and Temperature instead of only the supply voltage fluctuation.

$$K_{u\_\max}(t) = K_{u0}(t) + B_u(t)(V_{cc\_max} - V_{cc0}) + A_u(t)(V_{cc\_max} - V_{cc0})^2$$

$$K_{u\_\min}(t) = K_{u0}(t) + B_u(t)(V_{cc\_\min} - V_{cc0}) + A_u(t)(V_{cc\_\min} - V_{cc0})^2$$

$$K_u(t)*I_u(V_1) + K_d(t)*I_d(V_1) = I_{out}(V_1)$$

$$K_u(t)*I_u(V_2) + K_d(t)*I_d(V_2) = I_{out}(V_2)$$

- I-V data already provided in IBIS
- Related to process corner

## Solution

- Consider DC jitter sensitivity when calculating Ku/Kd for cases with the non-nominal supply voltage.
- Introduction of PSIJ keyword is needed.

# Jitter Sensitivity Based Modification

- Jitter sensitivity can be applied to calculate the total jitter

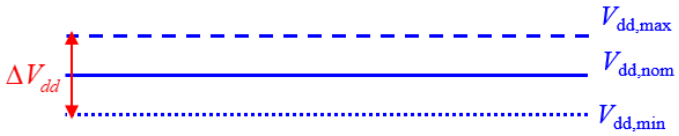
$$\text{Jitter Impact}(f) = \text{Jitter Sensitivity}(f) \cdot V_{noise}(f)$$

$$\text{Jitter Sensitivity}(\omega) = \frac{T_{pd\,\max} - T_{pd\,\min}}{V_{dd\,\max} - V_{dd\,\min}} PSRR'(\omega) e^{j\pi f T_{p0}} \sin c(\pi f T_{p0})$$

*Jitter sensitivity @ DC

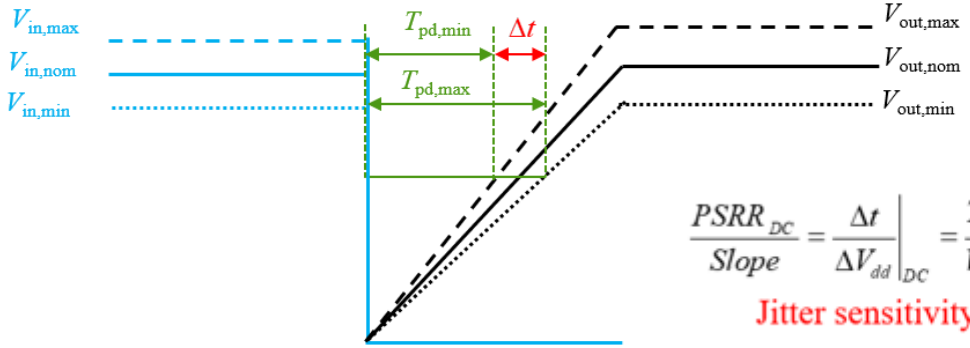
Frequency dependency due to time averaged effect (already considered by average the Vcc)

Frequency dependency due to PSRR (Power Supply Rejection Ratio)




- Propose to use Jitter sensitivity to do modification

$$\text{Ku/Kd}_{max/min}(t) = \text{Ku/Kd}_{typ}(t \pm \text{Jitter sensitivity} \times \Delta\text{Vdd})$$



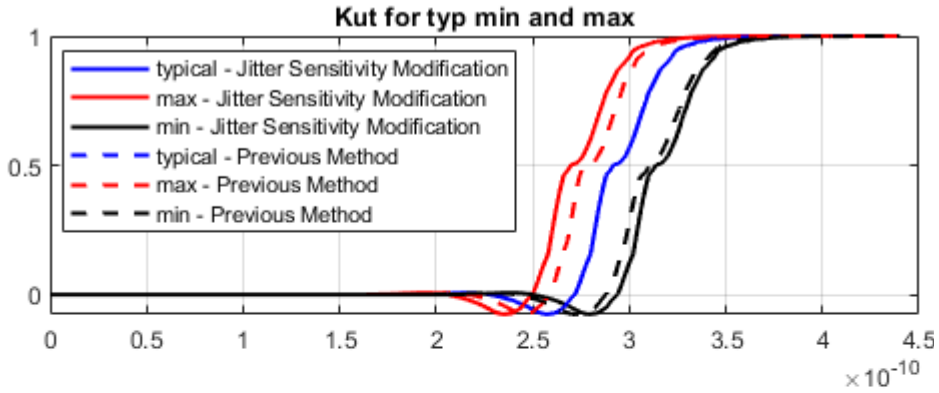

$$\frac{PSRR_{DC}}{Slope} = \left.\frac{\Delta t}{\Delta V_{dd}}\right|_{DC} = \frac{T_{pd\,\max} - T_{pd\,\min}}{V_{dd\,\max} - V_{dd\,\min}}$$

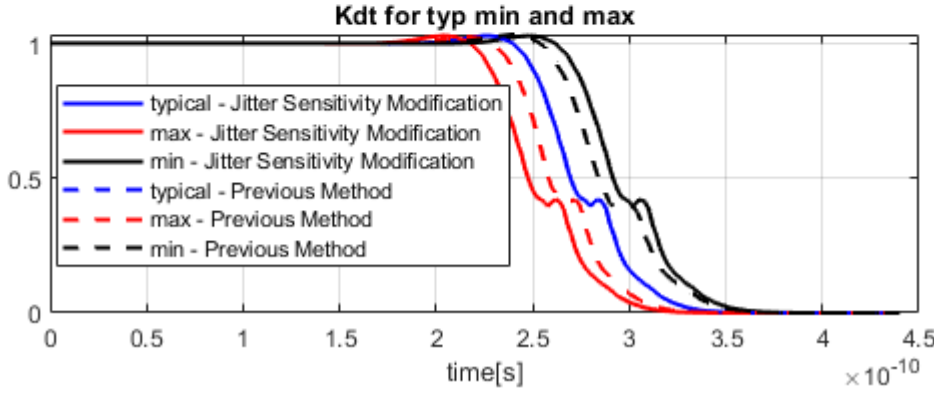
Jitter sensitivity @ DC

*Y. Sun, J. Lee and C. Hwang, "A Generalized Power Supply Induced Jitter Model Based on Power Supply Rejection Ratio Response," in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 29, no. 6, pp. 1052-1060, June 2021, doi: 10.1109/TVLSI.2021.3072799.*

# Jitter Sensitivity Based Modification

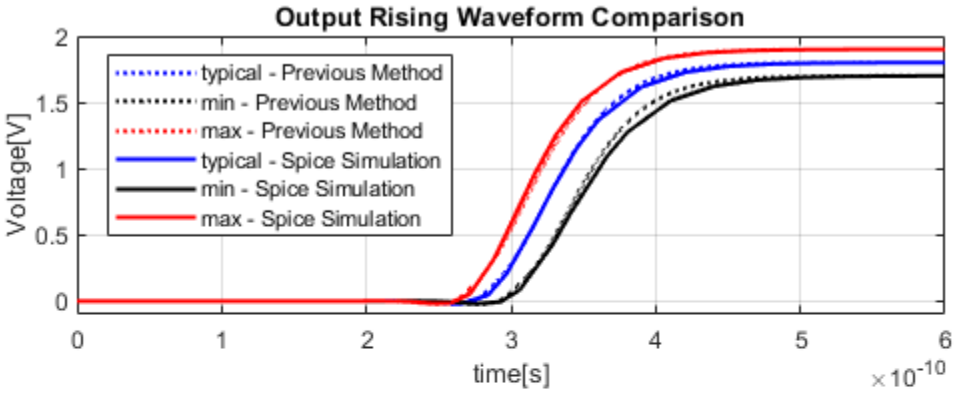
- Ku/Kd comparison for two methods

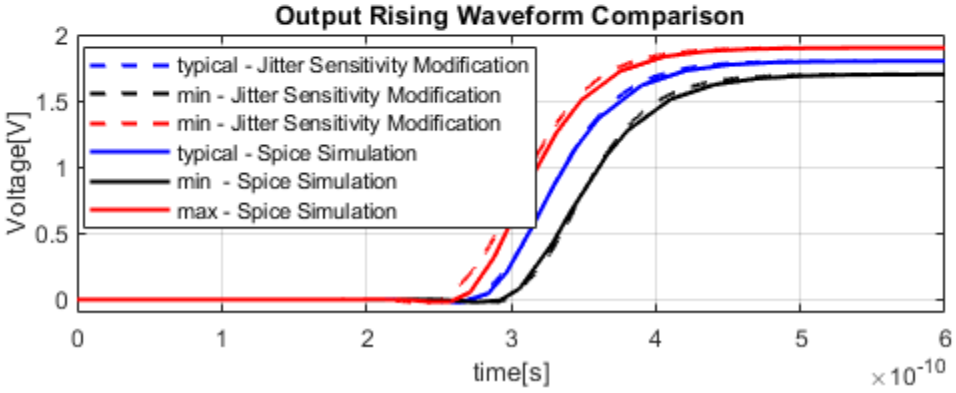



- Jitter sensitivity for this validation case is 206.7ps/V
- The Ku/Kd and output for "Previous Method" mentioned here is after Ku/Kd correction in the previous slides.

- Output comparison for two methods and Spice




- Rising edge jitter from Spice simulation is 44.5ps
- Rising edge jitter for previous method is 34.15ps (23.26%)
- Rising edge jitter for jitter sensitivity modification method is 45.82ps (2.97%)

# Applied to Over-clocking Case

## Feedback 3

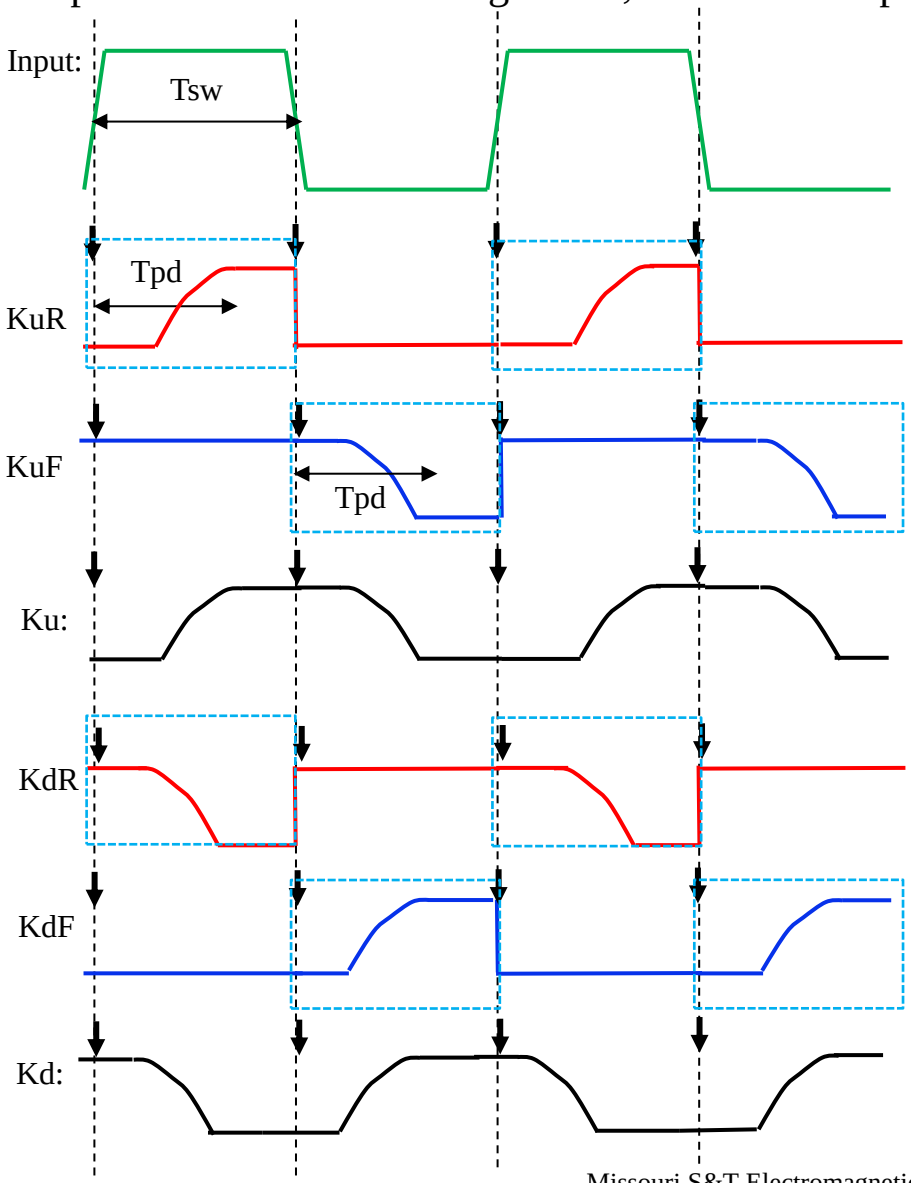
- Algorithm is for the case that driver propagation delay is smaller than the input switching period. Need to consider the over-clocking cases.

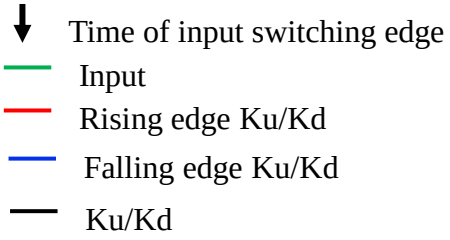
## Solution

- Use more delay elements to store value of rising and falling switching time and averaged Vcc.
- Set Ku/Kd tuning logic for different stages properly case by case.

# Previous Modification

- In previous modification algorithm, for the case Tpd < Tsw



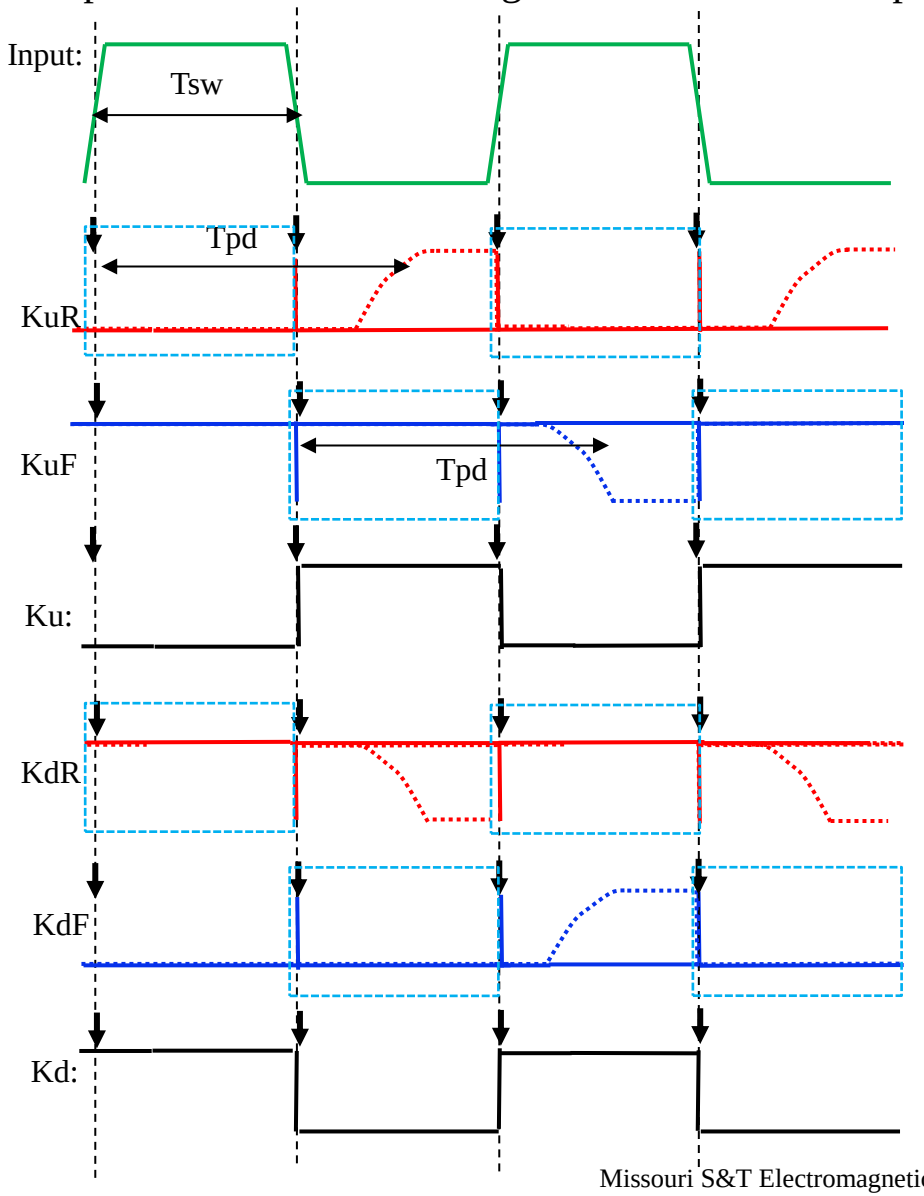

Previous modification methods:

- Introduce delay element to store the time elapsed since the switching.
- Introduce element to store the time averaged Vcc since input switching happens.

- KuR, KdR, KuF, KdF change when the input changes.
- Ku/Kd for the whole waveform is the combination of Ku/Kd for rising and falling edge.

# Previous Modification

- In previous modification algorithm, for the case Tpd > Tsw




Time of input switching edge
Input
Simulated Rising edge Ku/Kd
Simulated Falling edge Ku/Kd
Actual Rising edge Ku/Kd
Actual Falling edge Ku/Kd
Ku/Kd

Problem:

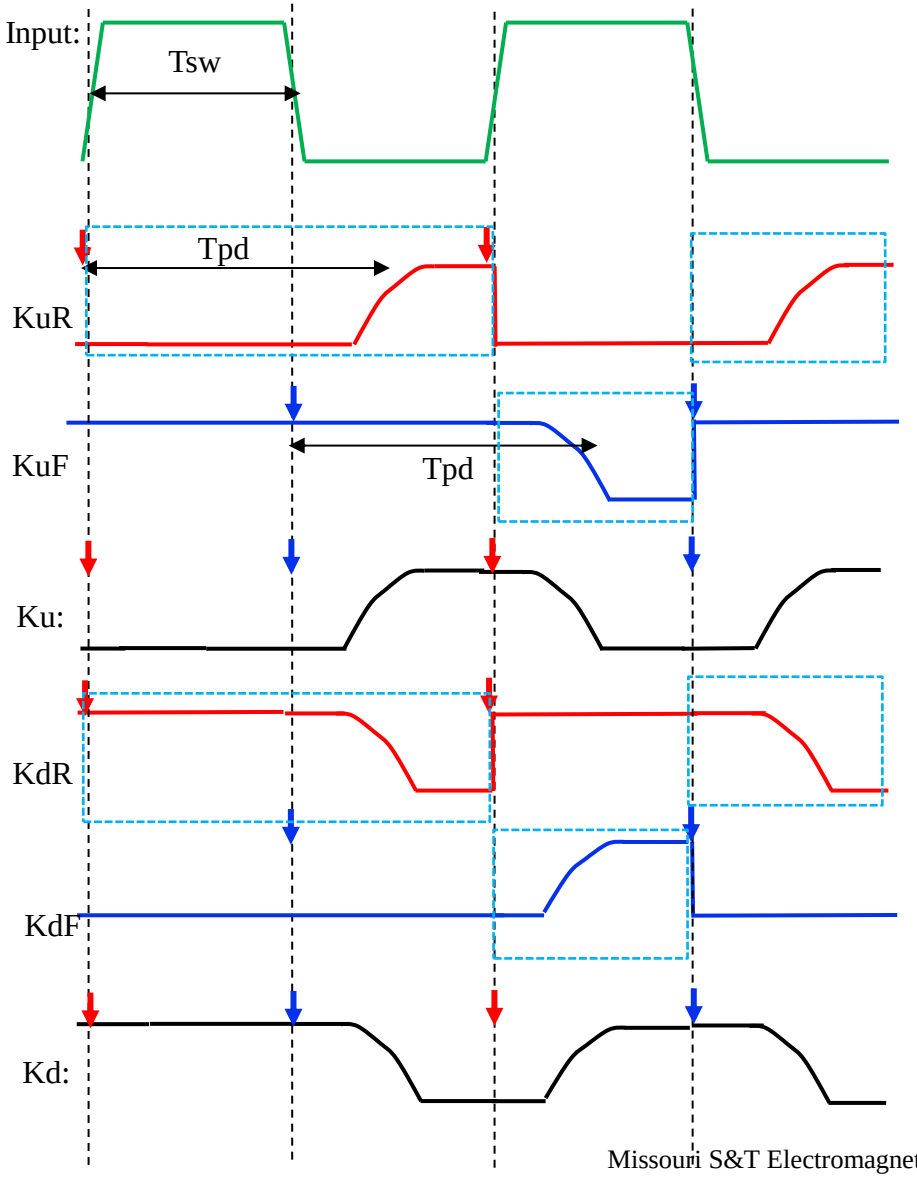
- KuR, KdR, KuF, KdF change when the input changes.
- KuR, KdR, KuF, KdF cannot have switching behavior before the input switching due to the longer propagation delay.
- The combined Ku/Kd is incorrect.

⇩

Consider rising edge and falling edge separately

# Applied to Over-clocking Case

- In new proposed algorithm, for the case Tpd > Tsw




Time of input rising edge

Time of input falling edge

Input

Rising edge Ku/Kd

Falling edge Ku/Kd

Ku/Kd

New Modification:

- Introduce delay element to store the time elapsed since the rising and falling switching respectively.

- KuR and KdR change at the input rising edge.
- KuF and KdF change at the input falling edge.

- Ku/Kd for the whole waveform is also the proper combination of Ku/Kd for rising and falling edge.

# Applied to Over-clocking Case

- Implementation in Ngspice (Modify based on current ibis2spice algorithm)
  Use more elements to store value of rising and falling switching time and averaged Vcc
  (**Improved algorithm in this work, a practical implementation in open-source Ngspice**)

```
* INPUT CONTROL
BN NINX 0 V= ((V(NINP) > 0.0) && (V(NENB) > 0.5))? 1.0 : 0.0

* CONTROL LOGIC
BI NI 0 V=(V(NINX) - 0.5)
B2 N2 0 V=V(NI, N9) * 8
B3 N3 0 V=abs(V(N2))
B4 N4 0 V=(V(N3) > 0.5)? 1 : -1
B51 N51 0 V=(V(N2) > 0.5)? TIME * 1E9: 0
B52 N52 0 V=(V(N2) < -0.5)? TIME * 1E9: 0
B61 N61 0 V=(V(N2) > 0.5)? V(N51): V(N81)
B62 N62 0 V=(V(N2) < -0.5)? V(N52): V(N82)
B71 NX1 0 V=(V(N61) >= 1.0)? TIME * 1E9 - V(N81) : 0.0
B72 NX2 0 V=(V(N62) >= 1.0)? TIME * 1E9 - V(N82) : 0.0
B81 NT11 0 V=(V(NX1) > 0.01)? (V(NVCC)*0.001 -1.8*0.001 +V(NTD1)): 0.0
B82 NT21 0 V=(V(NX2) > 0.01)? (V(NVCC)*0.001 -1.8*0.001 +V(NTD2)): 0.0
B91 NT12 0 V=(V(NX1) > 0.01)? V(NT11)/V(NX1): 0.0
B92 NT22 0 V=(V(NX2) > 0.01)? V(NT21)/V(NX2): 0.0

* DELAY ELEMENT: Td value must match time-step
T11 N61 0 N81 0 Z0=50 Td=1p
T12 N62 0 N82 0 Z0=50 Td=1p
T2 NI 0 N9 0 Z0=50 Td=1p
T31 NT11 0 NTD1 0 Z0=50 Td=1p
T32 NT21 0 NTD2 0 Z0=50 Td=1p
R11 N81 0 50
R12 N82 0 50
R2 N9 0 50
R31 NTD1 0 50
R32 NTD2 0 50
```

- V(NX1): Time elapsed since input rising switching event happens
- V(NX2): Time elapsed since input falling switching even happens

- V(NT11): Accumulated voltage since input rising switching event happens
- V(NT21): Accumulated voltage since input falling switching event happens

- V(NT12): Time averaged Vcc since input rising switching event happens
- V(NT22): Time averaged Vcc since input falling switching event happens

# Applied to Over-clocking Case

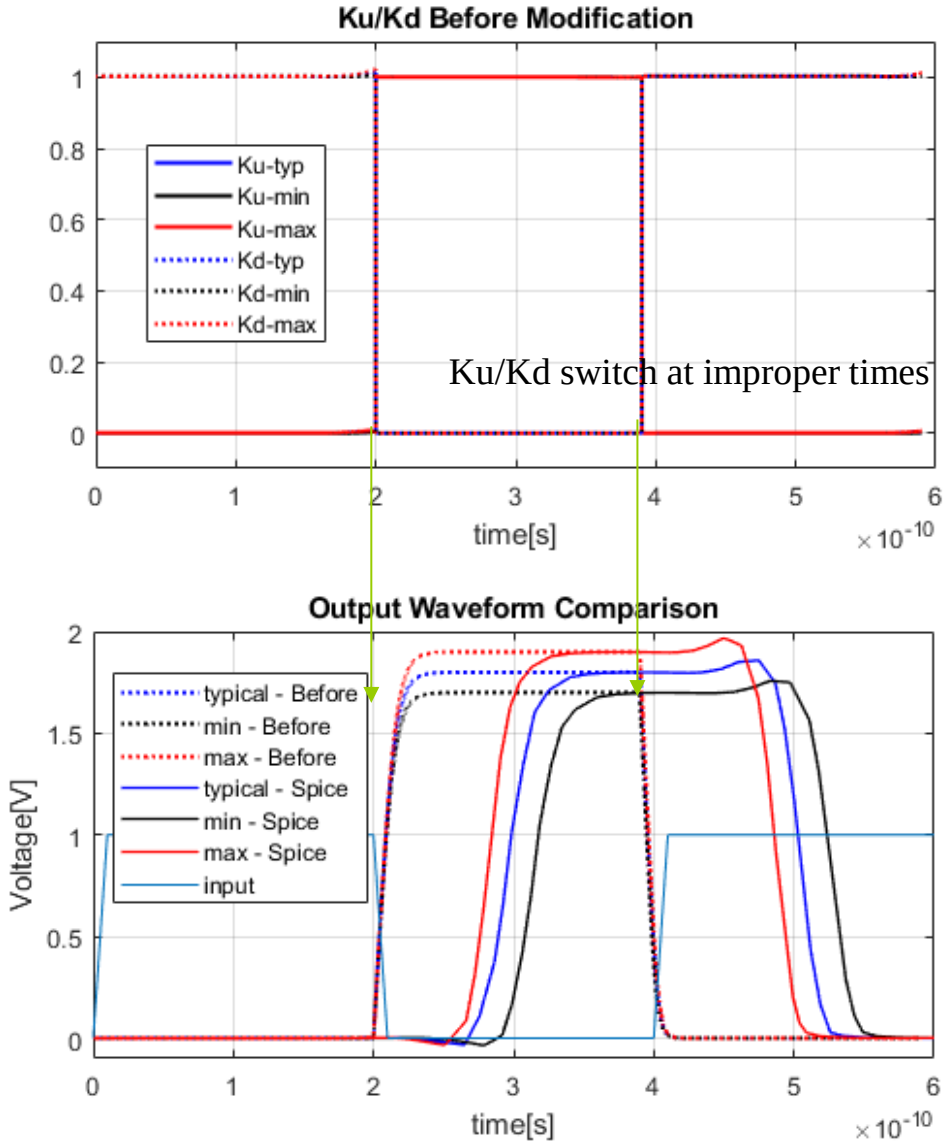
- Implementation in Ngspice (Modify based on current ibis2spice algorithm)
  Edit Ku/Kd tuning logic (**Improved algorithm in this work, a practical implementation in open-source Ngspice**)

```
* KU/KD COEF.
XASRC_KUR NKUR0 0 NX1 0 NT12 driver2_TYP_KU_R
XASRC_KDR NKDR0 0 NX1 0 NT12 driver2_TYP_KD_R
XASRC_KUF NKUF0 0 NX2 0 NT22 driver2_TYP_KU_F
XASRC_KDF NKDF0 0 NX2 0 NT22 driver2_TYP_KD_F

* KU/KD TUNING
BKUF NKUF 0 V = (V(N62) > 0.5) ? (TIME*1E9 >=1 && TIME*1E9-V(N62) > 0 )? V(NKUF0): 1 : 1
BKDF NKDF 0 V = (V(N62) > 0.5) ? (TIME*1E9 >=1 && TIME*1E9-V(N62) > 0 )? V(NKDF0): 0 : 0
BKUR NKUR 0 V = (V(N61) > 0.5) ? (TIME*1E9 >=1 && TIME*1E9-V(N61) > 0 )? V(NKUR0): 0 : 0
BKDR NKDR 0 V = (V(N61) > 0.5) ? (TIME*1E9 >=1 && TIME*1E9-V(N61) > 0 )? V(NKDR0): 1 : 1

BU   NKUX 0 V =
+(V(N62) > V(N61))? V(NKUR):
+(V(N61) > V(N62) && V(N62) >0.5)?    V(NKUF):
+ 0.0

BD   NKDX 0 V =
+(V(N62) > V(N61))? V(NKDR):
+(V(N61) > V(N62) && V(N62) >0.5)?    V(NKDF):
+ 1.0
```

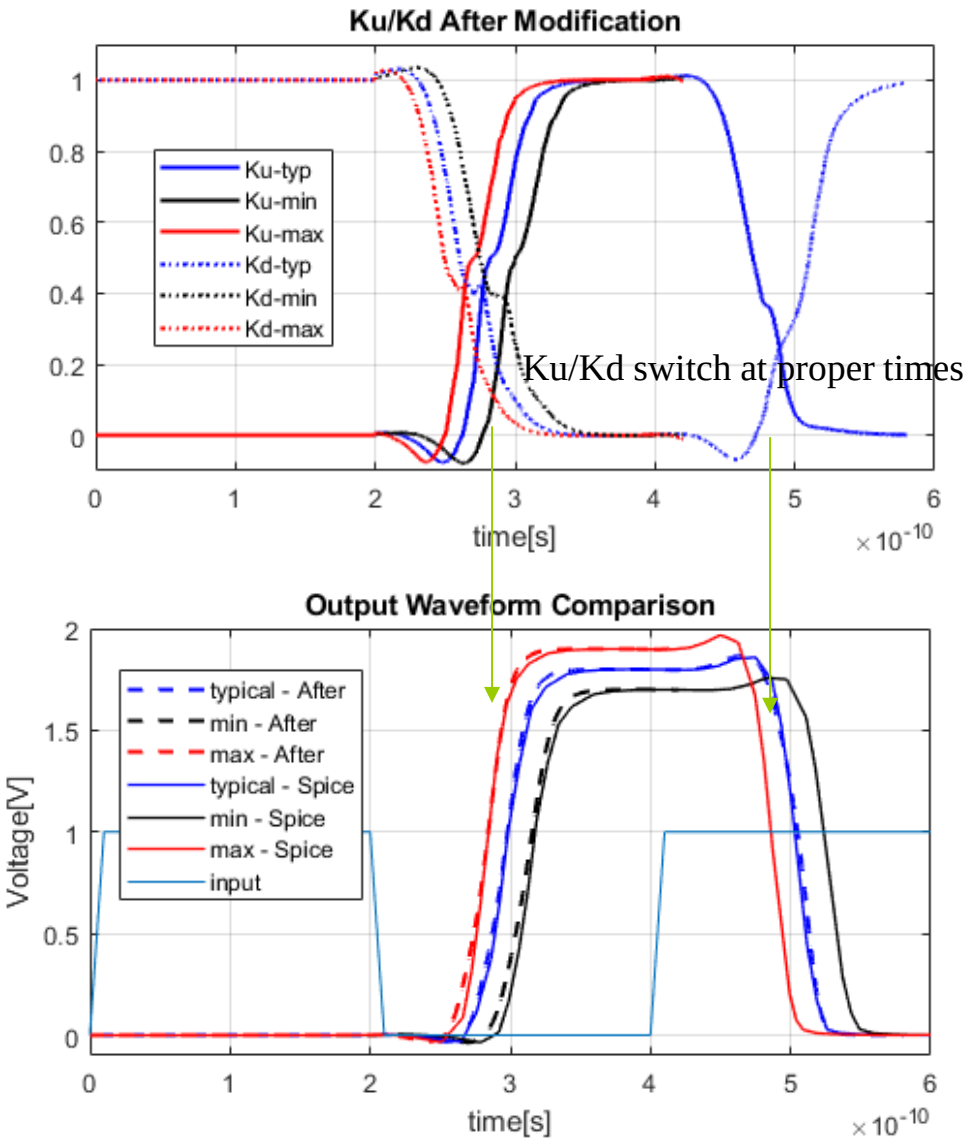
*Properly combine Ku and Kd for different stages

# Applied to Over-clocking Case

## Before Modification




## After Modification



# Conclusions

- The accuracy of original Ku/Kd modification-based IBIS simulation has been improved.
- A new modification method based on PSIJ sensitivity is proposed.
- This modification algorithm can be extended to the over-clocking cases.

# Thanks for Listening

# Model Implementation

- Implementation in Ngspice (Modify based on current ibis2spice algorithm)
  Implement the time averaged Vcc (**<u>Improved algorithm in this work, a practical implementation in open-source Ngspice</u>**)

```
* INPUT CONTROL
BN NINX 0 V= ((V(NINP) > 0.5) & (V(NENB) > 0.5))? 1.0 : 0.0

* CONTROL LOGIC
BI NI 0 V=(V(NINX) - 0.5)
B2 N2 0 V=V(NI, N9) * 8
B3 N3 0 V=abs(V(N2))
B4 N4 0 V=(V(N3) > 0.5)? 1 : -1
B5 N5 0 V=V(N4) > 0? TIME * 1E9: 0
B6 N6 0 V=V(N4) > 0? V(N5) : V(N8)
B7 NX 0 V=(V(N6) >= 1.0)? TIME * 1E9 - V(N8) : 0.0
B8 NT1 0 V=(V(NX) > 0.01)? (V(NVCC)*0.001-1.8*0.001+V(NTD)) : 0.0
B9 NT 0 V=(V(NX) > 0.01)? V(NT1)/V(NX) : 0.0

* DELAY ELEMENT: Td value must match time-step
T1 N6 0 N8 0 Z0=50 Td=1p
T2 NI 0 N9 0 Z0=50 Td=1p
T3 NT1 0 NTD 0 Z0=50 Td=1p
R1 N8 0 50
R2 N9 0 50
R3 NTD 0 50
```

Vcc

Vcc0

NT1 NTD

ideal transmission line

V(NT1) store the summation of Vcc voltage since start of switching

Realized by:
Vcc-Vcc0+V(NTD)

V(NX) time elapsed since the switching

**V(NT)** is the time averaged Vcc

$$\frac{\int_0^t V_{cc}(\tau)d\tau}{t}$$

# Model Implementation

- Implementation in Ngspice (Modify based on current ibis2spice algorithm)
  Implement the modified Ku, Kd as B source (**Improved algorithm in this work, a practical implementation in open-source Ngspice**)

Original Ku implementation: Ku0(t)

```
* KU COEF RISE
.SUBCKT driver_TYP_KU_R 3 4 1 2
B1 3 4 V =
+ (V(1,2) < 0.000000E0)? 0.000000E0:
+ (V(1,2) < 3.622352E-3)? 1.287944E1 * V(1,2) + 0.000000E0:
+ (V(1,2) < 7.244704E-3)? -7.295161E-5 * V(1,2) + 4.665411E-2:
```

Ku0(t)

Modified Ku implementation

```
* KU COEF RISE
.SUBCKT driver_TYP_KU_R 3 4 1 2 5
B1 3 4 V =
+ (V(1,2) < 0.000000E0)? 0.000000E0:
+ (V(1,2) < 0.0036223520000000)? 12.879440000000007 * V(1,2) + 0.0000000000000000
+ (V(1,2) < 0.0072447040000000)? -0.0000729516100000 * V(1,2) + 0.0466541100000000

  + (0.0000251111959497 * V(1,2) + -0.1034584500000000) * V(5) + (0.0039680452540881 * V(1,2) + -7.5443674999999999) * V(5) * V(5) :
  + (0.0002022823036612 * V(1,2) + -0.1034590917761164) * V(5) + (0.0091645843101826 * V(1,2) + -7.5443863236936428) * V(5) * V(5) :
```

Bu(t)

Au(t)

B source in Ngspice to store tabulated data

**5.1 Bxxxx: Nonlinear dependent source (ASRC)**

**5.1.1 Syntax and usage**

General form:

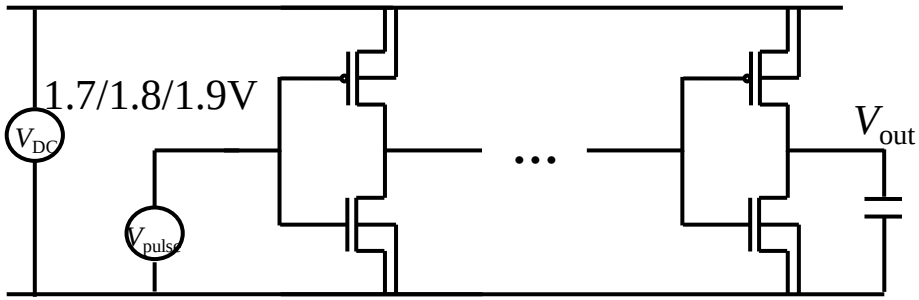
```
BXXXXXXX n+ n- <i=expr> <v=expr> <tc1=value> <tc2=value>
+ <temp=value> <dtemp=value>
```

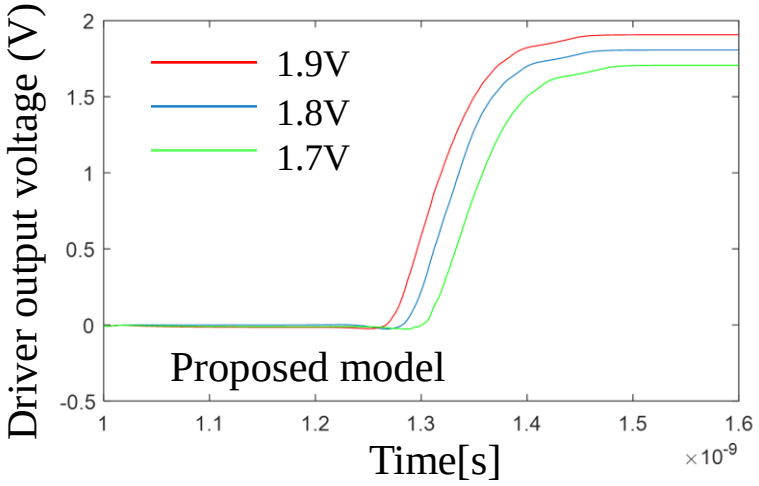
Examples:

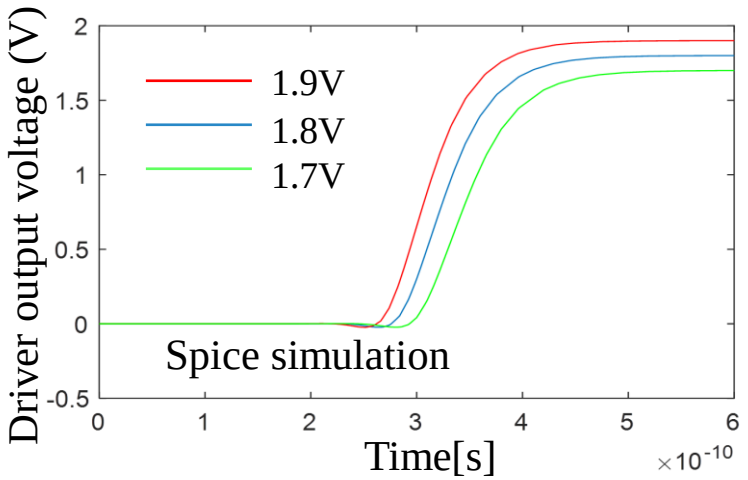
```
B1 0 1 I=cos(v(1))+sin(v(2))
B2 0 1 V=ln(cos(log(v(1,2)^2)))-v(3)^4+v(2)^v(1)
B3 3 4 I=17
B4 3 4 V=exp(pi^i(vdd))
B5 2 0 V = V(1) < {Vlow} ? {Vlow} : V(1) > {Vhigh} ? {Vhigh} : V(1)
```

# Model Validation

1. Vcc 1.7/1.8/1.9V respectively

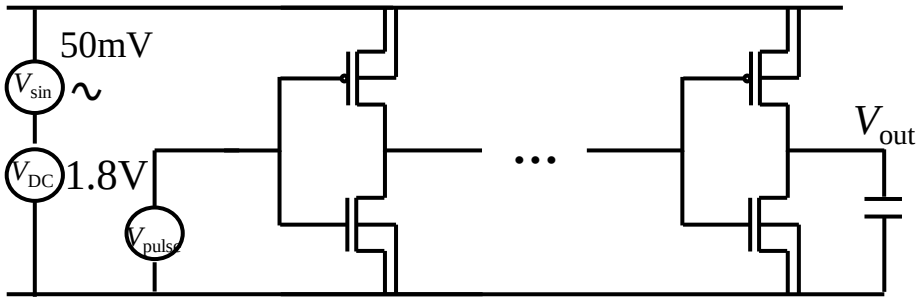


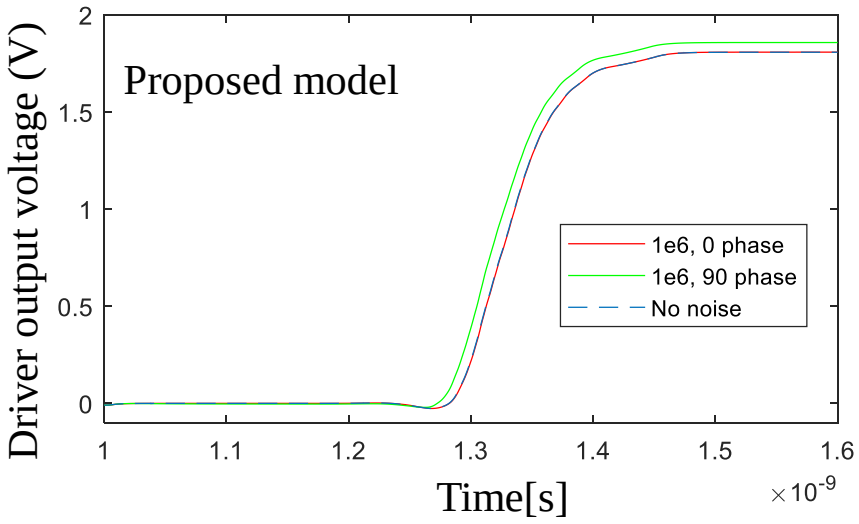
# Model Validation

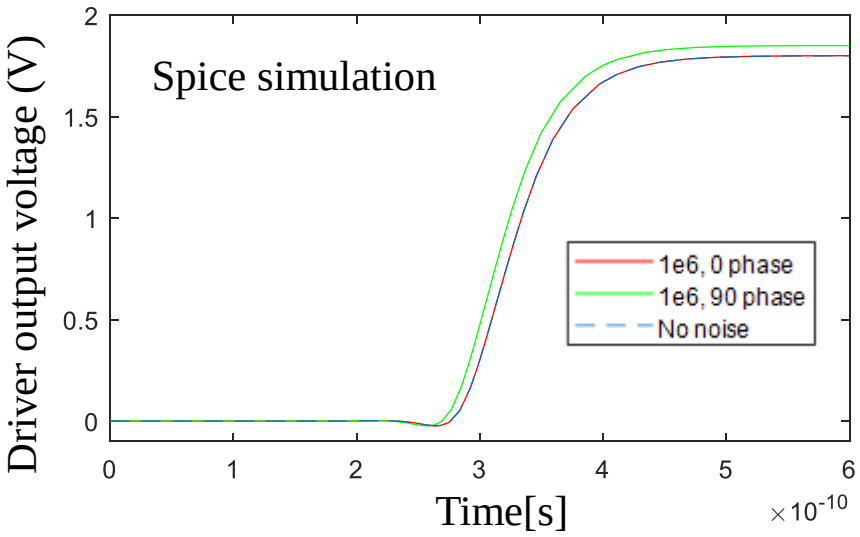
2. Vcc have very low frequency noise




Vcc=1.8V+0.05*sin(2*pi*1e6)

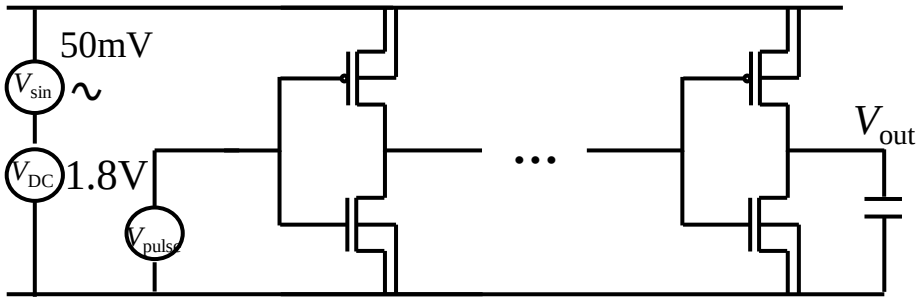
Vcc=1.8V+0.05*sin(2*pi*1e6+pi/2)



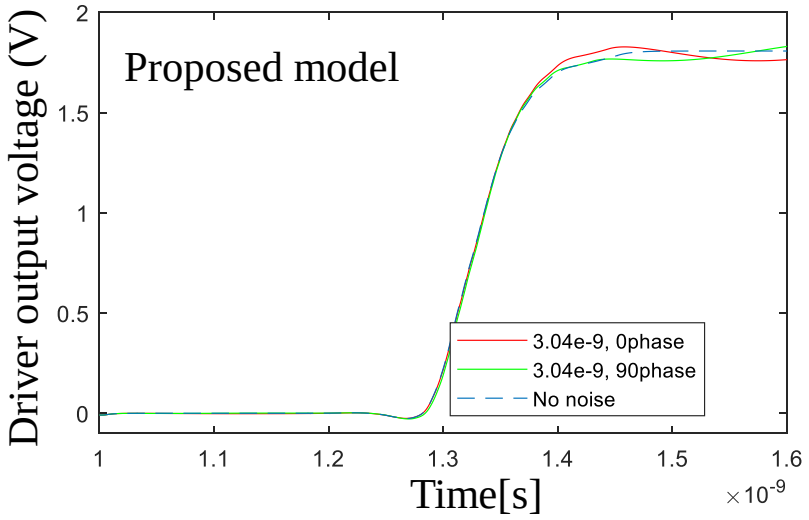
# Model Validation

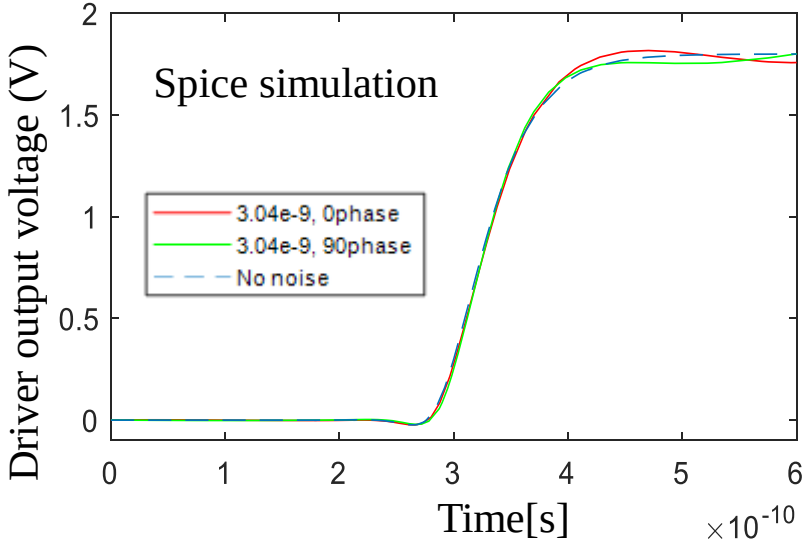
3. Vcc have noise with frequency corresponds to propagation delay (329ps)




Vcc=1.8V+0.05*sin(2*pi*3.04e9)

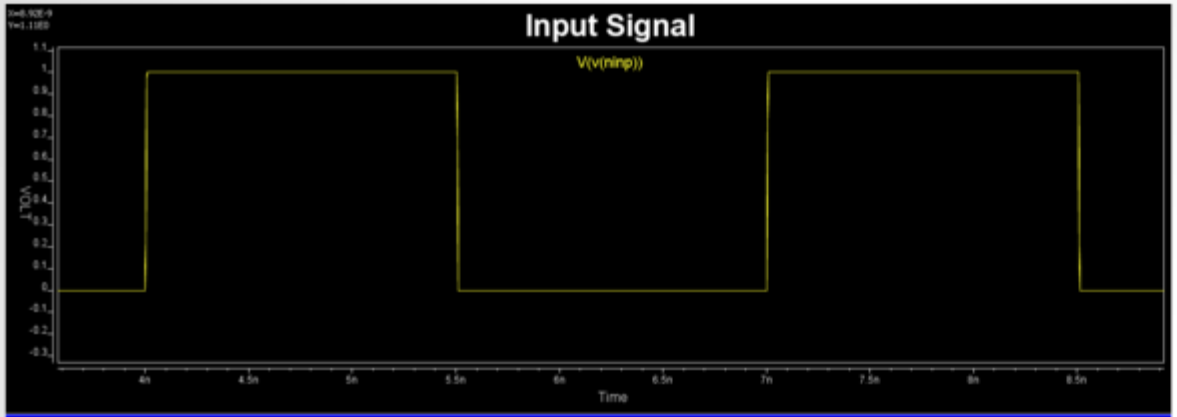
Vcc=1.8V+0.05*sin(2*pi*3.04e9+pi/2)



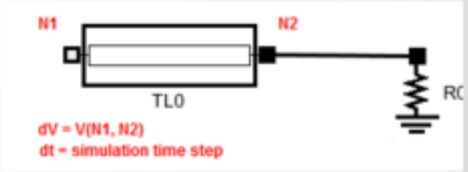
# Implementation of New Behavior Model Proposal

- Implementation in Ngspice (Modify based on current ibis2spice algorithm)
  1. Ku, Kd, Bu, Au, Bd, Ad calculated offline from rising/falling waveforms
  2. From input switching edge dv/dt, judging rising or falling




Source:
http://www.spisim.com/blog/ibis2spice_p1/
http://www.spisim.com/blog/ibis2spice_p2/

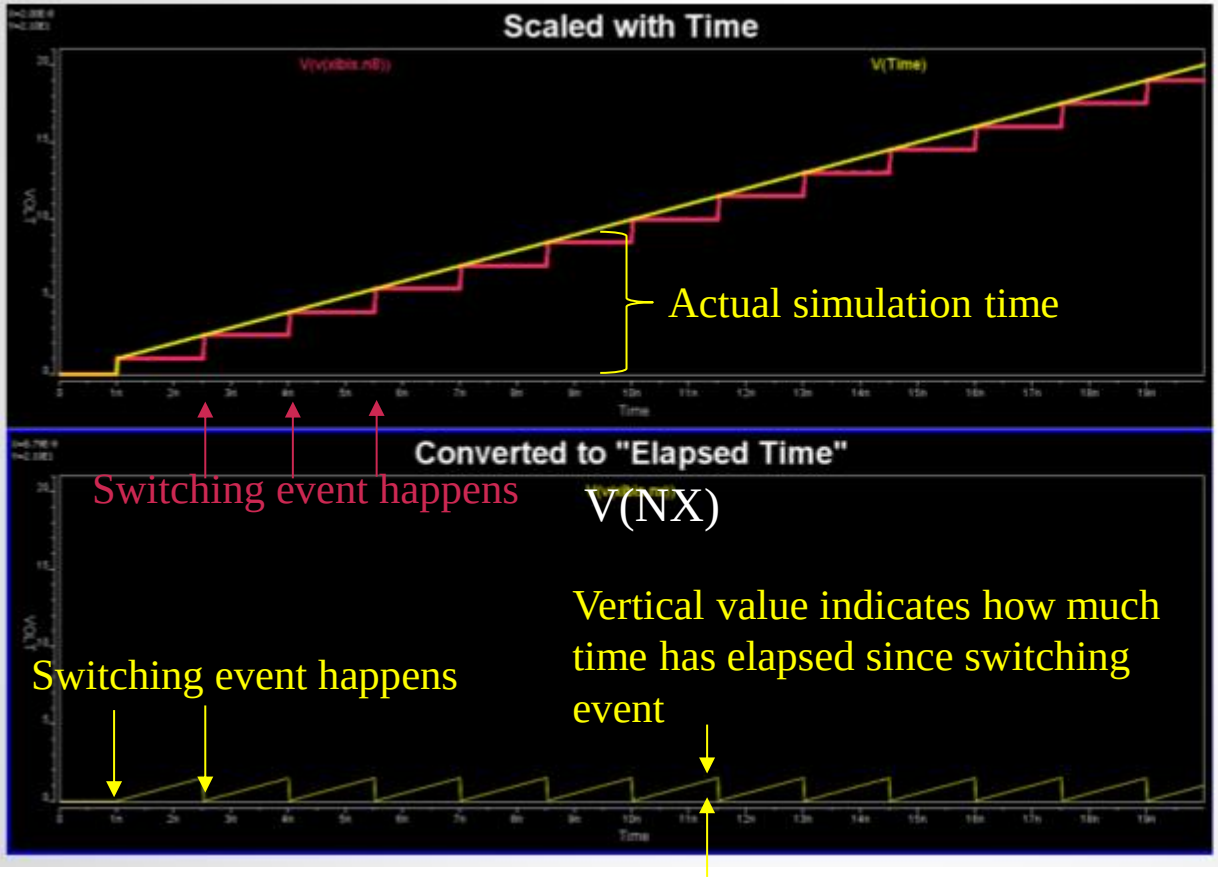
Use a transmission line to realize the differentiation



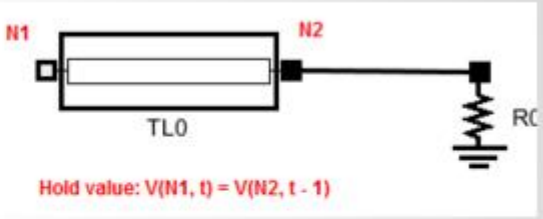
# Implementation of New Behavior Model Proposal

- Implementation in Ngspice (Modify based on current ibis2spice algorithm)

  3. Record elapsed time since every switching event




Source:
http://www.spisim.com/blog/ibis2spice_p1/
http://www.spisim.com/blog/ibis2spice_p2/

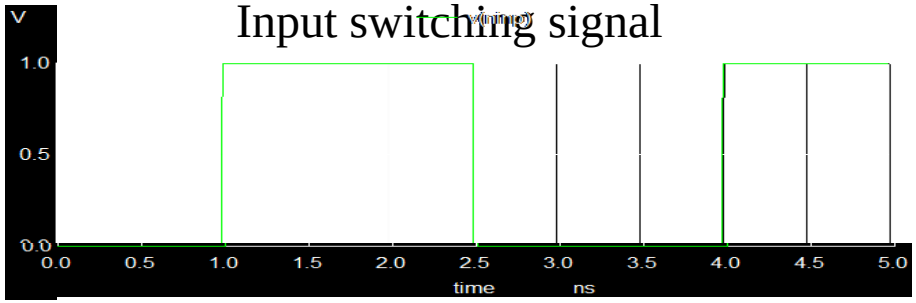


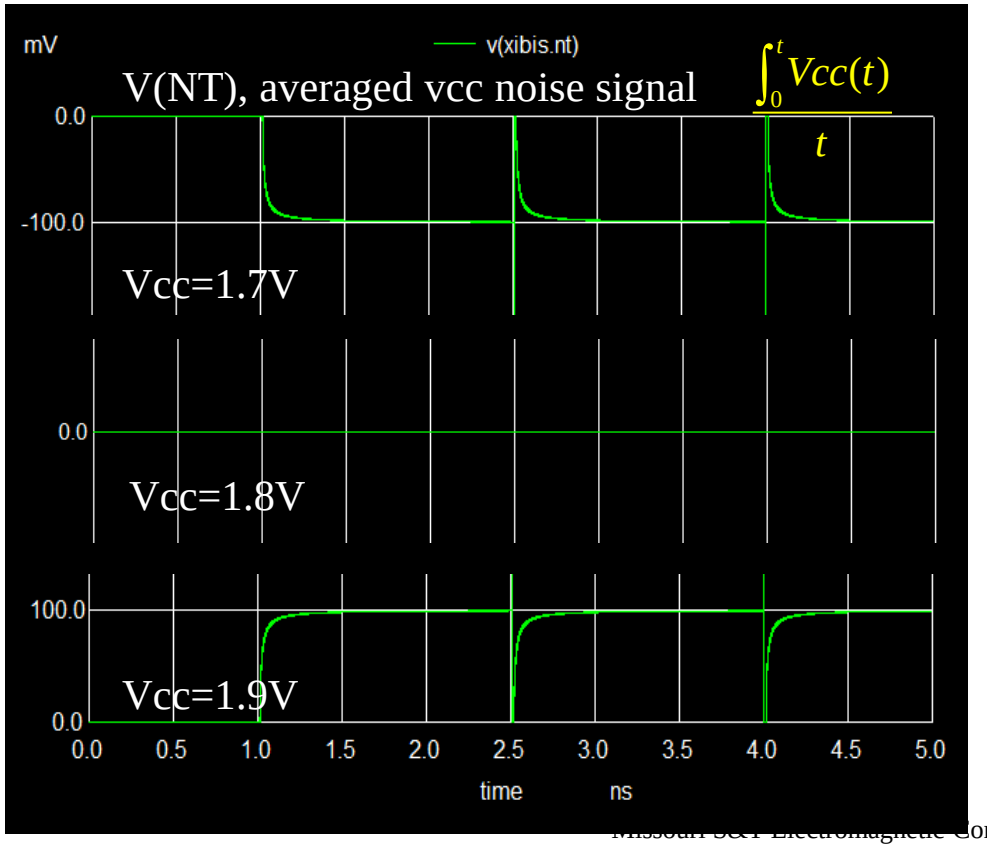

The level hold (latch) realized with an ideal transmission line

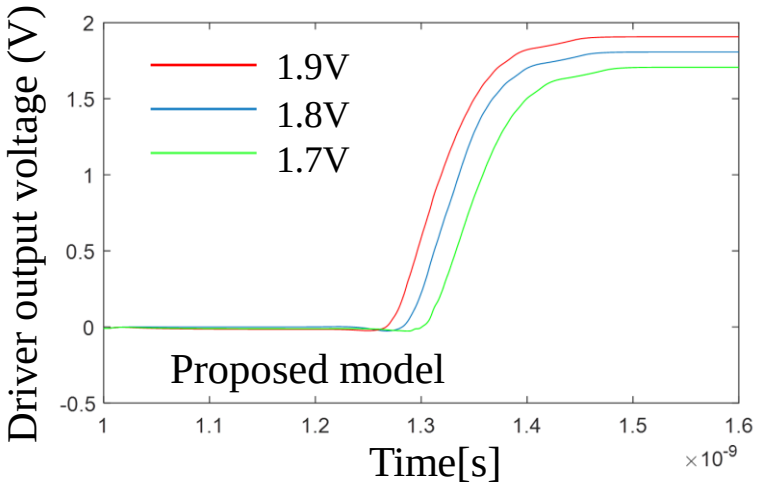
t - value hold

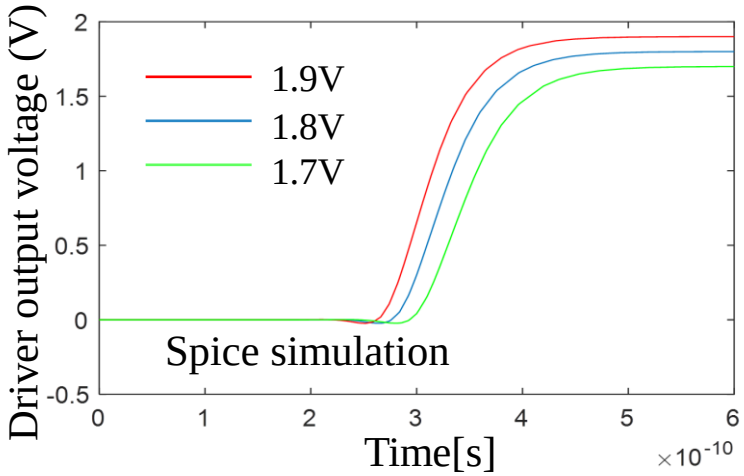
# Simulation Results of Implemented Time-Averaged Vcc[V(NT)]

1. Vcc 1.7/1.8/1.9V respectively



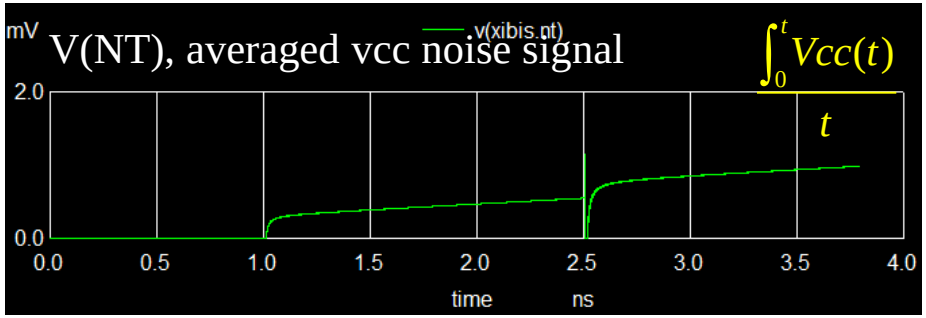



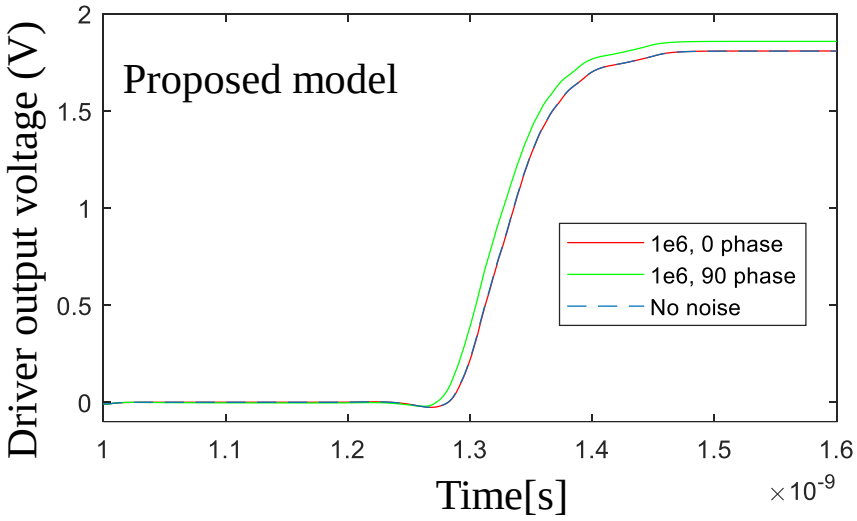



# Simulation Results of Implemented Time-Averaged Vcc[V(NT)]

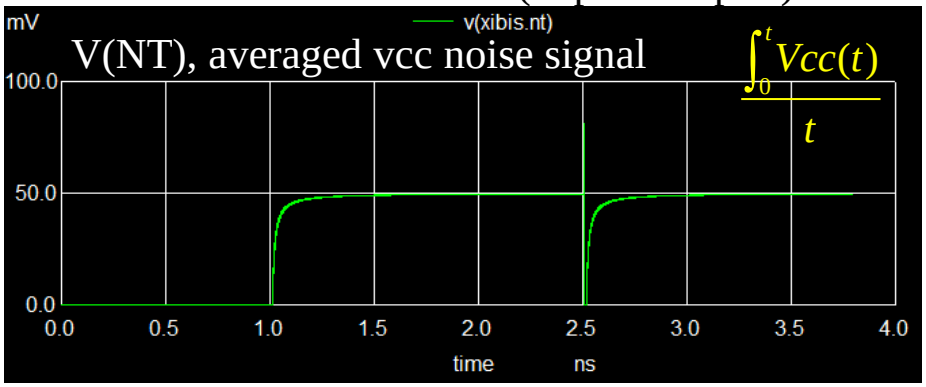
2. Vcc have very low frequency noise

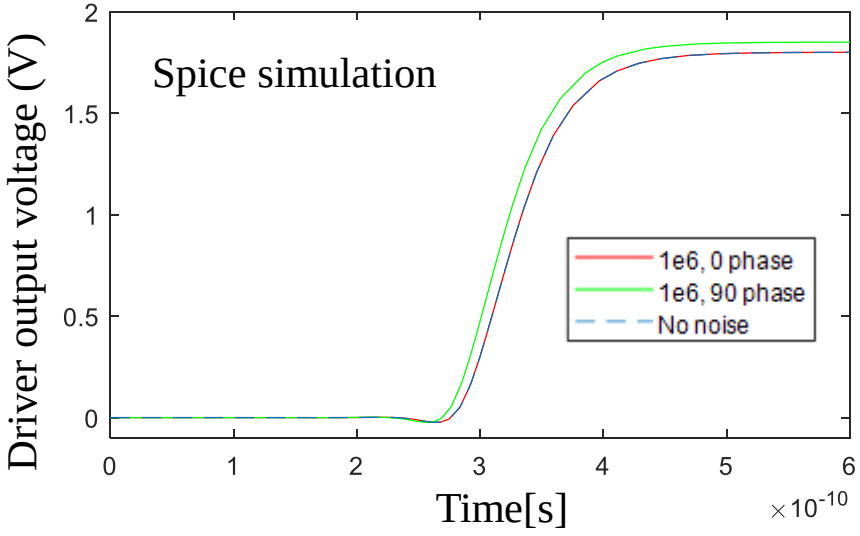
Vcc=1.8V+0.05*sin(2*pi*1e6)




Vcc=1.8V+0.05*sin(2*pi*1e6+pi/2)

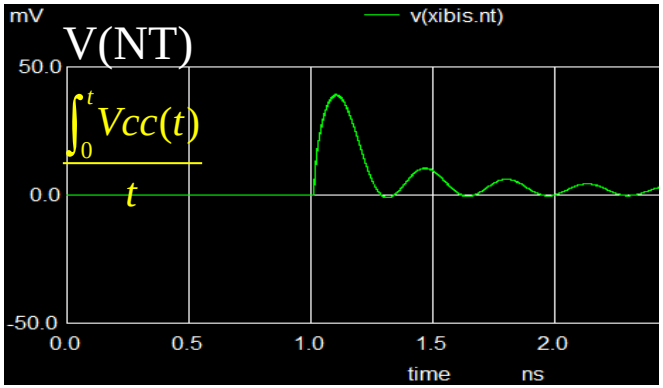


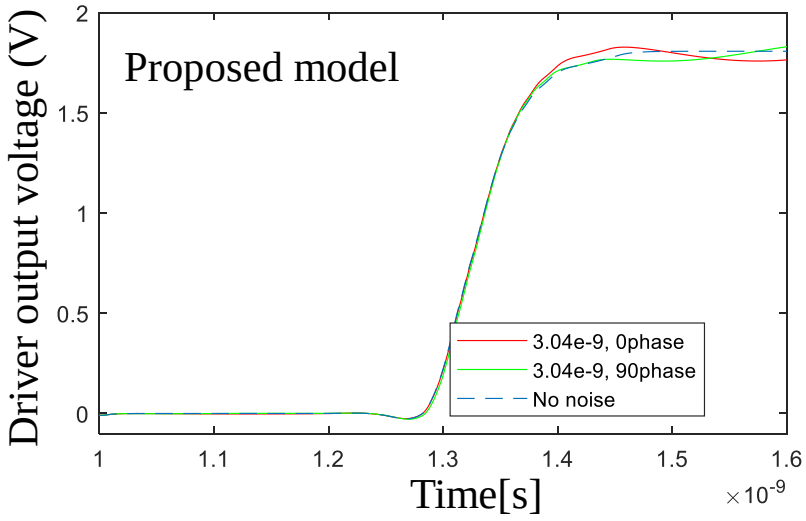



# Simulation Results of Implemented Time-Averaged Vcc[V(NT)]

3. Vcc have noise with frequency corresponds to propagation delay (329ps)

Vcc=1.8V+0.05*sin(2*pi*3.04e9)




Vcc=1.8V+0.05*sin(2*pi*3.04e9+pi/2)