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.

\[ V_{in} \rightarrow \text{Power supply noise} \rightarrow V_{dd} \]

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

\[ V_{in} \rightarrow V_{out} \]

\[ V_{in} \rightarrow T_{p0} \rightarrow V_{out} \]
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
(ter5.1, 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

\[
K_d(t)I_{pd} \rightarrow K_{sspdc}(V_{pd})K_d(t)I_{pd}
\]

\[
K_u(t)I_{pu} \rightarrow K_{sspu}(V_{pu})K_u(t)I_{pu}
\]

\[
K_{sspdc}(V_{pd}) = \frac{V_{pd}}{I_{sspdc}(0)}
\]

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

- The ratio modification Ksspdc, 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 \( K_u(t), K_d(t) \) as a function of \textbf{time averaged} power rail voltage \( V_{cc}(t) \);

- Introduce correction coefficients \( B \) and \( A \) as a function of \textbf{time}.

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

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

- 2 equations, 2 unknowns' algorithm to extract \( K_u(t), K_d(t) \) achieved by adding delay elements that store
  - The time of switching edges
  - Time averaged \( V_{cc} \) since switching event happens
**Previous Proposed Model Validation**

- Tested driver

![Circuit Diagram](image)

180nm technology, nominal voltage 1.8V

**Output Rising Waveform Comparison**

- **Kut for typ min and max**
  - typical
  - min
  - max

- **Kdt for typ min and max**
  - typical
  - min
  - max

- **Output Rising Waveform Comparison**
  - typical - Previous Method
  - min - Previous Method
  - max - Previous Method
  - typical - Spice Simulation
  - min - Spice Simulation
  - max - Spice Simulation

Missouri S&T Electromagnetic Compatibility Laboratory
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

\[ K_u(t) \times I_u(V_1) + K_d(t) \times I_d(V_1) = I_{out}(V_1) \]
\[ K_u(t) \times I_u(V_2) + K_d(t) \times I_d(V_2) = I_{out}(V_2) \]

- 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

<table>
<thead>
<tr>
<th>Voltage at Reference</th>
<th>Simulation Tool 1</th>
<th>Simulation Tool 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>PU IV typ</td>
<td>-1.4mA@refV</td>
<td>17.47uA@refV</td>
</tr>
<tr>
<td>PU IV min</td>
<td>0.7mA@refV</td>
<td>19.04uA@refV</td>
</tr>
<tr>
<td>PU IV max</td>
<td>1.7mA@refV</td>
<td>15.95uA@refV</td>
</tr>
<tr>
<td>PD IV typ</td>
<td>2.5mA@refV</td>
<td>-8.29uA@refV</td>
</tr>
<tr>
<td>PD IV min</td>
<td>1mA@refV</td>
<td>-12.15uA@refV</td>
</tr>
<tr>
<td>PD IV max</td>
<td>-3mA@refV</td>
<td>-4.92uA@refV</td>
</tr>
</tbody>
</table>

- 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.

\[
K_{u_{\text{max}}}(t) = K_{u0}(t) + B_u(t)(V_{cc_{\text{max}}} - V_{cc0}) + A_u(t)(V_{cc_{\text{max}}} - V_{cc0})^2
\]

\[
K_{u_{\text{min}}}(t) = K_{u0}(t) + B_u(t)(V_{cc_{\text{min}}} - V_{cc0}) + A_u(t)(V_{cc_{\text{min}}} - V_{cc0})^2
\]

\[
K_u(t)I_u(V_1) + K_d(t)I_d(V_1) = I_{out}(V_1)
\]

- 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

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

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

*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

\[
\frac{Ku/Kd_{\text{max/min}}(t)}{Ku/Kd_{\text{typ}}(t \pm \text{Jitter sensitivity} \times \Delta V_{dd})}
\]

Jitter Sensitivity Based Modification

- Ku/Kd comparison for two methods

- Output comparison for two methods and Spice

- 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.

- 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 $T_{pd} < T_{sw}$

Previous modification methods:
• Introduce delay element to store the time elapsed since the switching.
• Introduce element to store the time averaged $V_{cc}$ since input switching happens.

• $K_{uR}$, $K_{dR}$, $K_{uF}$, $K_{dF}$ change when the input changes.
• $K_u/K_d$ for the whole waveform is the combination of $K_u/K_d$ for rising and falling edge.
Previous Modification

- In previous modification algorithm, for the case $T_{pd} > T_{sw}$

Problem:
- $Ku_R$, $Kd_R$, $Ku_F$, $Kd_F$ change when the input changes.
- $Ku_R$, $Kd_R$, $Ku_F$, $Kd_F$ 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.
In new proposed algorithm, for the case $T_{pd} > T_{sw}$

**New Modification:**
- Introduce delay element to store the time elapsed since the rising and falling switching respectively.
- $K_{uR}$ and $K_{dR}$ change at the input rising edge.
- $K_{uF}$ and $K_{dF}$ change at the input falling edge.
- $K_u/K_d$ for the whole waveform is also the proper combination of $K_u/K_d$ 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)

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

* CONTROL LOGIC
B1 N1 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

* DELAY ELEMENT: Td value must match time-step
T11 N61 0 N81 0 Z=50 Td=1p
T12 N62 0 N62 0 Z=50 Td=1p
T2 N1 0 N9 0 Z=50 Td=1p
T3 T11 0 NTD1 0 Z=50 Td=1p
T32 T21 0 NTD2 0 Z=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 event 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

Ku/Kd switch at improper times

After Modification

Ku/Kd switch at proper times

Before Modification

After Modification

Output Waveform Comparison

Output Waveform Comparison
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 (**Improved algorithm in this work, a practical implementation in open-source Ngspice**)

\[
\begin{align*}
V(NT1) & \text{ store the summation of } Vcc \\
V(NX) & \text{ time elapsed since the switching}
\end{align*}
\]

**Vcc**

Vcc-Vcc0+V(NTD)

\[
\mathbf{V(NT)} \text{ is the time averaged } Vcc \\
\frac{\int_0^t V_{cc}(\tau)d\tau}{t}
\]
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)

```
* KU COEF RISE
.SUBCKT driver_TYP_KU_R 3 4 1 2
BU3 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:
```

Original Ku implementation: Ku0(t)

```
* KU COEF RISE
.SUBCKT driver_TYP_KU_R 3 4 1 2 5
BU3 3 4 V =
+ (V(1,2) < 0.000000E0)? 0.000000E0:
+ (V(1,2) < 0.0036223520000000)? 12.879440000000000 * V(1,2) + 0.0000000000000000:
+ (V(1,2) < 0.0072447040000000)? -0.00000729516100000 * V(1,2) + 0.046654110000000:
+ (0.000025111959497 * V(1,2) + -0.10345845000000000) * 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) + -7.544386326936428) * V(5) * V(5):
```

Modified Ku implementation

```
BU(t)
Au(t)
```

5.1 Bxxxx: Nonlinear dependent source (ASRC)

5.1.1 Syntax and usage

General form:

```
Bxxxxxx 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)^22)))-v(3)^4+v(2)^4 * v(1)
B3 3 4 i=17
B4 3 4 v=exp(pi*i(vdd))
```
Model Validation

1. Vcc 1.7/1.8/1.9V respectively

Driver output voltage (V)

![Proposed model graph](image1)

![Spice simulation graph](image2)
2. Vcc have very low frequency noise

\[ V_{\text{out}} = V_{\text{DC}} + 50 \text{mV} \sin(2\pi \times 10^6 t) \]

\[ V_{\text{cc}} = 1.8V + 0.05 \sin(2\pi \times 10^6 t + \pi/2) \]

**Proposed model**

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

\[ V_{cc} = 1.8V + 0.05 \sin(2\pi \times 3.04e9) \]

\[ V_{cc} = 1.8V + 0.05 \sin(2\pi \times 3.04e9 + \pi/2) \]
Implementation of New Behavior Model Proposal

- Implementation in Ngspice (Modify based on current ibis2spice algorithm)
  1. $K_u, K_d, B_u, A_u, B_d, A_d$ 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

![Diagram showing time scaling and switching events](image)

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

Input switching signal

\[
V(NT), \text{averaged } Vcc \text{ noise signal } \int_0^t Vcc(t)
\]

Driver output voltage (V)

- Vcc=1.7V
- Vcc=1.8V
- Vcc=1.9V

Time [s]

Spice simulation

Proposed model

Driver output voltage (V)

- 1.9V
- 1.8V
- 1.7V

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

2. Vcc have very low frequency noise

\[ V_{cc} = 1.8V + 0.05\sin(2\pi \times 1e6) \]

\[ V_{cc} = 1.8V + 0.05\sin(2\pi \times 1e6 + \pi/2) \]

![Graph showing proposed model and Spice simulation](image)
Simulation Results of Implemented Time-Averaged Vcc[V(NT)]

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

\[ V_{cc} = 1.8V + 0.05\sin(2\pi \times 3.04\times10^9) \]

\[ V_{cc} = 1.8V + 0.05\sin(2\pi \times 3.04\times10^9 + \pi/2) \]