Synopsys®
CCS noise Characterization Guideline

Version 1.4, October 2016

SYNOPSYS®

Contents

Introduction

This document provides techniques to create accurate CCS noise libraries. The first section describes simulation techniques and measurements to characterize CCS noise library data using an inverter as an example. The second section describes CCS noise characterization for multi-stage cells. CCS noise library examples are provided for combinational cells and macro cells.

CCS noise Characterization of a Channel-Connected Block (CCB)

The following section describes how to select characterization waveforms, select table indexes, and generate Liberty syntax for one channel connected block (CCB). An inverter is used as the example.

CCS noise DC Current Table

To characterize the DC current table, apply an input voltage source at stage input and an output voltage source at stage output. Sweep the voltage values at both input and output sides. Record the DC currents through the output voltage source.

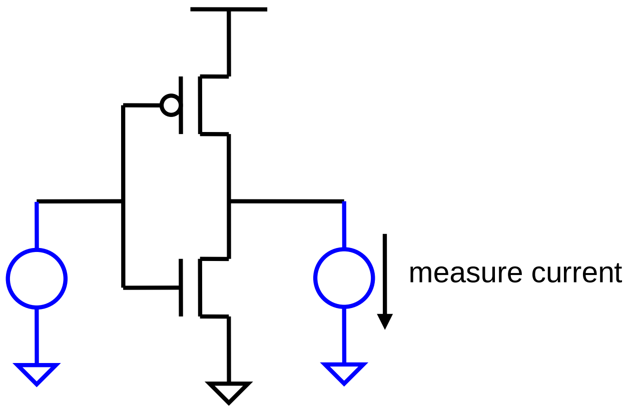


Figure 1 Measuring DC currents through output voltage source

For best accuracy and coverage of the voltage coverage, you can use a table with 29 by 29 indexes, where the normalized voltage index values are:

```
-1.0, -0.5, -0.2, -0.1, 0.0, 0.05, … (one per 0.05) …, 0.95, 1.0, 1.1,
1.2, 1.5, 2.0
```

Note that the voltage values in the library are in library voltage unit and are not normalized to VDD.

Valid index_1 (input_voltage) is in increasing order. The first entry is less than or equal to VSS, and the last entry is more than or equal to VDD. Valid index_2 (output_voltage) is also in increasing order. The first entry is less than or equal to VSS, and the last entry is more than or equal to VDD. The current values stored in the table are recommended to have at least four significant digits for best accuracy.

The recommended range of characterization of the DC table refers to the absolute voltage. For voltage scaling in static timing and noise analysis, you must use the indices that correspond to the same percent of VDD in all libraries in one scaling group. An example of desirable index selection follows:

```
Lib1: nom_voltage=0.8:  0.00, 0.08, 0.16, 0.24, 0.32, ..., 0.80
Lib2: nom_voltage=1.0:  0.00, 0.10, 0.20, 0.30, 0.40, ..., 1.00
Lib3: nom_voltage=1.2:  0.00, 0.12, 0.24, 0.36, 0.48, ..., 1.20
```

CCS noise Stage Output Voltage Tables

This table contains timing information of a single CCB. Use ramp input waveforms to characterize the requested timing information. We recommend using two to four vectors (that is, two to four simulations). Note that excessive vectors based on a large number of input slew and load capacitance combinations is not necessary.

The accuracy of a CCS noise model is not very sensitive to the exact values of input slew and load capacitance that you choose. However, you can use the following simple scheme as a guideline for CCB timing table characterization:

- Use the second and middle index points of CCS timing table for input slew.
- If CCB output is an output pin, use the second and middle index points of the CCS timing table for load capacitance. If CCB output is an internal net, use a zero or close-to-zero load capacitance.

Based on the above guideline, the number of vectors or simulations needed for either rise or fall transition is as follows.

- If the CCB output is an output pin, then 4 vectors or simulations are needed.
- If the CCB output is an internal node, then 2 vectors or simulations are needed.

For each vector or simulation, it is recommended to use 5 time and voltage pairs. This is measured when the output voltage waveform crosses 10 percent, 30 percent, 50 percent, 70 percent, and 90 percent of VDD. The time reference is the start of the input ramp waveform, as illustrated in the figure below. Please note that this is different from CCS timing delay measurement where the reference time is the delay trip point of the input waveform.

For CCS noise characterization, the transition time of the input ramp is measured from rail to rail. This is different from CCS timing tables where the transition times are measured between slew trip-points.

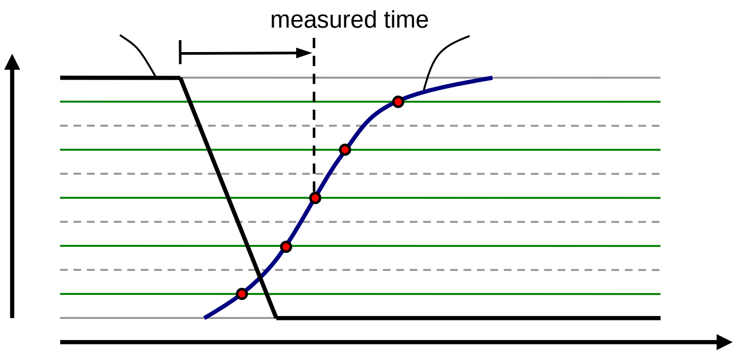


Figure 2 Measuring output voltage waveform

Valid input_transition is greater than or equal to 0. Valid net_capacitance is greater than or equal to 0. Valid index_3 (time) is a sequence of floating-point numbers in increasing order. Valid voltage values are between 0 (inclusive) and VDD (inclusive). The output voltage must transition to full rail voltage. If this condition is not met, it is likely due to incorrect CCB creation.

The voltage waveform measurement is used instead of current waveform measurement because the latter is much more difficult to measure on cell internal nodes with detailed parasitics. An example of such a node is shown in the following figure, where the first CCB is characterized.

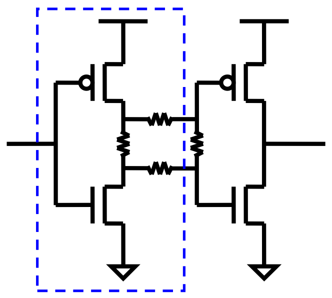


Figure 3 Measuring voltage waveform vs current waveform

CCS noise Propagation Tables

The CCS noise propagation table contains noise propagation information for a single CCB. Use symmetric triangular waveforms to characterize the required noise propagation information. There are three independent parameters, input noise height, input noise width, and load capacitance. Only sparse samples among the three-dimensional space are needed for CCS noise characterization. In general, the accuracy of the CCS noise model is not very sensitive to the exact values of noise height, width, and load capacitance that are being used to create the library.

As a guideline, you can use the second and the middle index points of the CCS timing table for total net capacitance, denoted as C_1 and C_2. As an exception, if CCB output is an internal node (that is, not an output pin of the cell), use zero or close-to-zero external load capacitance value as C_1. After a load capacitance is decided, choose appropriate input noise height and width combinations that will result to reasonable output noise bumps. One method to find such input noise height and width combinations is explained as follows.

1. Compute maximum drive current I_{MAX} for the CCB. For example, the maximum drive current for an inverter can be calculated as $I_{DC}(V_{DD}, V_{DD})$ for below_high noise region (with respect to cell output) and $I_{DC}(V_{SS}, V_{SS})$ for the above_low noise region, where $I_{DC}(\cdot)$ is the DC current as a function of input and output voltages. This function can be evaluated using 2-D interpolation of the DC current table available from previous DC characterization.
2. Find appropriate input noise height values. Taking below_high noise as an example, you can choose three input noise heights: H_1, H_2, and H_3, based on the following equations:

$$I_{DC}(H_1, V_{DD}) = 0.20 \cdot I_{MAX}$$
$$I_{DC}(H_2, V_{DD}) = 0.35 \cdot I_{MAX}$$
$$I_{DC}(H_3, V_{DD}) = 0.50 \cdot I_{MAX}$$

   where the second arguments in the $I_{DC}(\cdot)$ functions mean the CCB output is set to V_{DD}. The following figure illustrates this procedure.

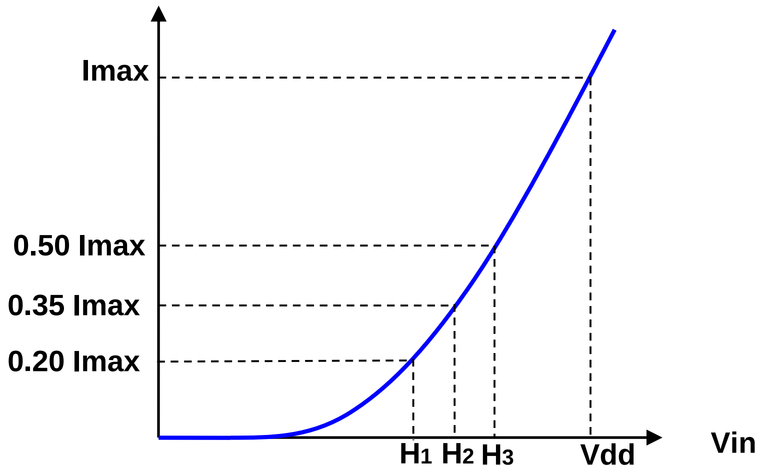


Figure 4 Computing input noise height values

3. Find appropriate noise width values that will result to reasonable propagated noise bumps. Using the transient timing simulation data obtained in stage output voltage tables, the amount of charge needed to produce ΔV voltage drop at the output node can be estimated as

$$Q(\Delta V) = I_{AVG}(V_{DD}) \cdot S + I_{MAX} \cdot (T(\Delta V) - S)$$

   where the average current is defined as

$$I_{AVG}(V) = \int_0^V I_{DC}(V_{in}, V_{DD}) \cdot dV_{in}$$

where the output voltage is fixed at its initial value V_{DD} for simplicity of calculation. When the input of the CCB is a triangular noise waveform, the noise width to produce similar amount of output voltage drop ΔV can be estimated as

$$W(V_P, \Delta V) = Q(\Delta V) / I_{AVG}(V_P)$$

where V_P is the peak voltage of the input noise.

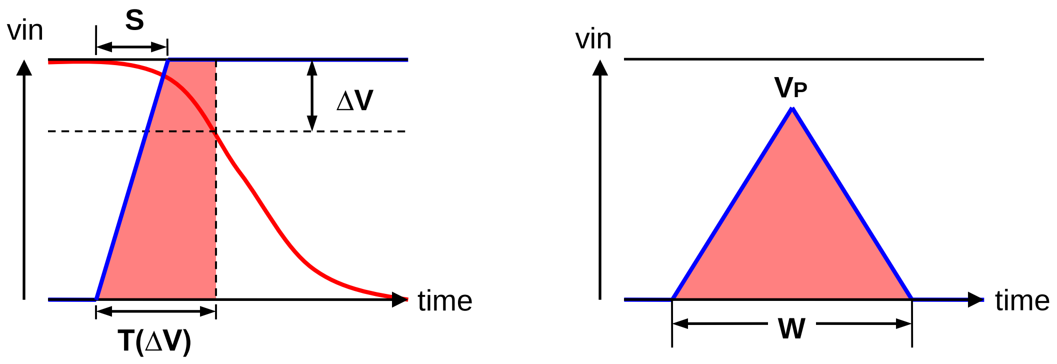


Figure 5 Estimating effective input area

4. Simulation vectors. If CCB output is an internal node of the cell, use the following 6 vectors for noise propagation simulations.

$$\begin{array}{lll} h = H_1 & w = W(H_1, \Delta_1) & c = C_1 \\ h = H_2 & w = W(H_2, \Delta_1) & c = C_1 \\ h = H_3 & w = W(H_3, \Delta_1) & c = C_1 \\ h = H_1 & w = W(H_1, \Delta_2) & c = C_1 \\ h = H_2 & w = W(H_2, \Delta_2) & c = C_1 \\ h = H_3 & w = W(H_3, \Delta_2) & c = C_1 \end{array}$$

where $\Delta_1 = 0.30 \cdot V_{DD}$ and $\Delta_2 = 0.50 \cdot V_{DD}$.

If CCB output is an output of cell, use the following two vectors in addition to the above 6 vectors.

$$\begin{array}{lll} h = H_2 & w = W(H_2, \Delta_2) & c = C_2 \\ h = H_3 & w = W(H_3, \Delta_2) & c = C_2 \end{array}$$

It is noted that the intention is to select a set of input noise bumps that produce reasonable and typical propagated noise bumps. Each propagated bump height should be at least 5% of VDD and at most 80% of VDD. In addition, propagated bump heights from all vectors should fully cover the typical range of 10-50% of VDD in a relatively evenly manner.

Example 1 Good table

Propagated noise bump heights are: 0.08, 0.12, 0.20, 0.25, 0.32, 0.40, 0.45, 0.55 (in VDD)

Example 2 Bad table as the first 3 noise bumps heights are too small

Propagated noise bump heights are: 0.02, 0.03, 0.04, 0.25, 0.32, 0.40, 0.45, 0.55 (in VDD)

Example 3 Bad table as there are too many noise bumps between 30-40%, but no coverage for 10-30% and 40-50%

Propagated noise bump heights are: 0.30, 0.32, 0.35, 0.35, 0.38, 0.38, 0.40, 0.40 (in VDD)

For each vector/simulation, store the following five points:

- First 50 percent bump height
- First 80 percent bump height
- Bump peak
- Fast 80 percent bump height
- Last 50 percent bump height

The time reference is the start of the rising edge of the input triangular waveform.

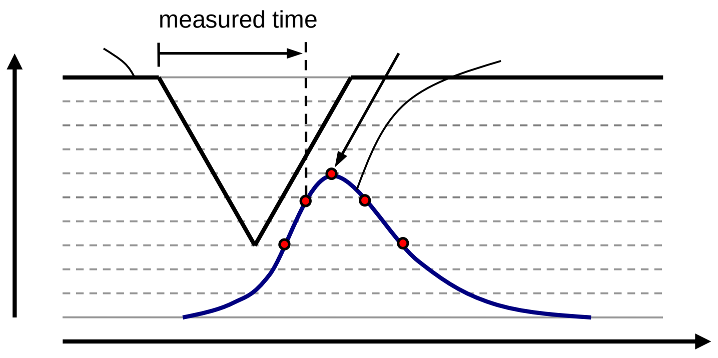


Figure 6 Measuring noise output waveform

The following describes the expected values for valid CCS noise library data.

- input_noise_height is from 0 (not inclusive) to VDD (inclusive).
- input_noise_width is greater than 0 (not inclusive).
- net_capacitance is greater than or equal to 0.
- index_4 (time) is floating-point numbers in increasing order.
- Voltage values are between 0 (inclusive) and VDD (inclusive).

Miller Capacitance Characterization

Miller capacitances can be characterized by performing SPICE simulations with designed setups. An example of such a setup is illustrated in Figure 7.

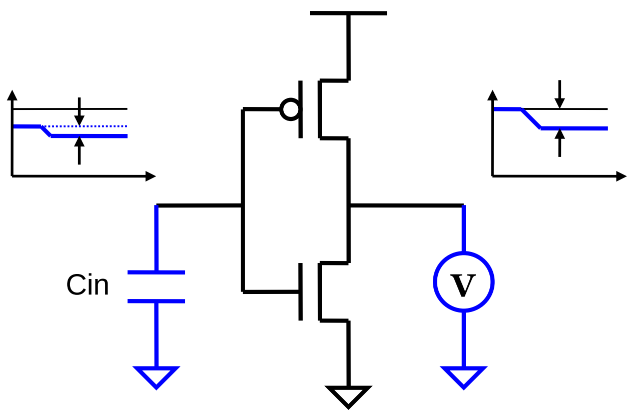


Figure 7 Measuring Miller Capacitance

A voltage source is applied at the output of the CCB and an external capacitance Cin is connected to the input of the CCB. Taking a falling output transition as an example, switch the output voltage source by ΔVout and measure the amount of voltage change at the input node ΔVin. Two simulations with same ΔVout but different Cin values (Cin,1 and Cin,2) can be performed. Denote the two measured input voltage changes ΔVin,1 and ΔVin,2.The Miller capacitance can be calculated using the following equation.

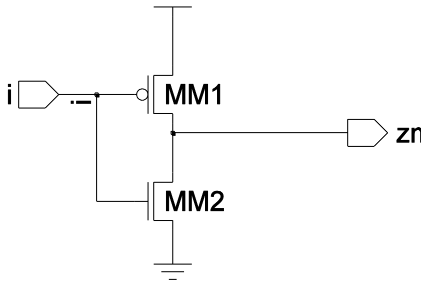
$$C_M = \frac{C_{in,1} - C_{in,2}}{\frac{\Delta V_{out}}{\Delta V_{in,1}} - \frac{\Delta V_{out}}{\Delta V_{in,2}}}$$

The two external capacitance values Cin,1 and Cin,2 can be chosen as 3 times and 10 times of the input pin capacitance. If input pin capacitance is not available, use the second and middle index points in the CCS timing table for total net capacitance.

With the FinFET technology, library Miller capacitance accuracy becomes more important for STA accuracy. Since Miller capacitances play the most significant role in timing waveform distortion due to receiver backward Miller effect, it is recommended to setup the initial voltage level of the input node and output voltage swing ΔVout to accurately model the average Miller capacitance for timing analysis backward Miller effect purpose. More specifically,

- Initial voltage of the input node should be set to a level near timing waveform tail, and
- Output voltage swing ΔVout should be large enough to model the full impact of backward Miller effect.

Liberty Syntax for Inverter Cell



```
/* Existing syntax library() group */
lu_table_template(del_0_5_7_t) {
 variable_1 : input_net_transition;
 variable_2 : total_output_net_capacitance;
}
/* New CCS syntax */
lu_table_template(ccsn_dc_10x10) {
 variable_1 : input_voltage;
 variable_2 : output_voltage;
}
lu_table_template(ccsn_timing_1d) {
  variable_1 : input_net_transition;
  variable_2 : total_output_net_capacitance;
  variable_3 : time;
}
lu_table_template(ccsn_prop_1d) {
  variable_1 : input_noise_height;
  variable_2 : input_noise_width;
  variable_3 : total_output_net_capacitance;
  variable_4 : time;
}


cell(inv0d0) {

  pin(I) {
    direction : input;
    …
  }

  pin(ZN) {
    direction : output;
    function : "(I)'";
    timing() {
      /* Existing syntax */
      related_pin   : "I";
      timing_sense  : negative_unate;
      cell_fall(del_0_5_7_t) {
        values( " … ");
      }
      fall_transition(del_0_5_7_t) {
        values( " … ");
      }
      cell_rise(del_0_5_7_t) {
        values( " … ");
      }
      rise_transition(del_0_5_7_t) {
        values( " … ");
      }
      /* New CCS syntax */
      ccsn_first_stage() {
          is_needed     : true;
          is_inverting    : true;
          stage_type      : both;
          miller_cap_rise : 0.00055;
          miller_cap_fall : 0.00084;

  dc_current (ccsn_dc_10x10) {
  index_1 (“0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0”);
  index_2 (“0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0”);
  values (“0.1, 0.1, 0.1, 0.1, 0.1, 0.2, 0.2, 0.2, 0.2, 0.2”, \
         “0.1, 0.1, 0.1, 0.1, 0.1, 0.2, 0.2, 0.2, 0.2, 0.2”, \
             …       …
         “0.1, 0.1, 0.1, 0.1, 0.1, 0.2, 0.2, 0.2, 0.2, 0.2”);
}
output_voltage_rise (  ) {
  vector (ccsn_timing_1d) {
    index_1 (“0.1”);
    index_2 (“0.1”);
    index_3 (“0.1, 0.2, 0.3”);
    values (“0.1, 0.2, 0.3”);
  }
  … …
}

output_voltage_fall (  ) {
  vector (ccsn_timing_1d) {
    index_1 (“0.1”);
    index_2 (“0.1”);
    index_3 (“0.1, 0.2, 0.3”);
    values (“0.1, 0.2, 0.3”);
  }
  … …
}
propagated_noise_low ( ) {
  vector (ccsn_prop_1d) {
    index_1 (“0.5”);
    index_2 (“1.0”);
    index_3 (“0.2”);
    index_4 (“0.1, 0.2, 0.3, 0.4, 0.5”);
    values (“0.0, 0.1, 0.2, 0.1, 0.0”);
  }
  … …
}

propagated_noise_high ( ) {
  vector (ccsn_prop_1d) {
    index_1 (“0.5”);
    index_2 (“1.0”);
    index_3 (“0.2”);
    index_4 (“0.1, 0.2, 0.3, 0.4, 0.5”);
    values (“1.0, 0.9, 0.8, 0.9, 1.0”);
  }
  … …
}

      }  /* end of ccsn_first_stage */
    } /* end of timing */
  } /* end of pin Z */
} /* cell(inv0d0) */
```

CCS noise Characterization of Cells with Multiple CCBs

This section describes CCS noise characterization of cells with multiple CCBs. Arc-based and pin-based CCS noise data will be discussed. In addition, CCB netlist partitioning, parasitic handling, and sensitization of input and output CCBs will be discussed.

Use of Arc-Based and Pin-Based CCS noise

Use the arc-based CCS whenever possible because it allows more detailed noise analysis, including noise propagation. Use the pin-based CCS noise in the following cases:

1. Arc describes a 3- or more-stage path.
2. Existence of a sensitizable 3- or more-stage path between the two arc pins. For example, if there are 2 parallel paths of lengths two and three, any unconditional CCS noise must be on pins. However, if there are two conditional timing arcs that strictly sensitize the two paths individually, the arc that describes the two-stage path can contain the CCS noise. The 3-stage path must always be described by pin-based CCS noise.
3. Electrical reconvergence somewhere between the the arc from and to pins. By electrical reconvergence is meant that signals causing rising and falling output propagate along two independent paths (see Section 0). Similarly to the above, if only one of the paths is sensitizable under a condition, the conditional arc-based CCS noise can be used.

Both arc- and pin-based CCS noise groups can be combined on the same cell. Generally, if a path is described by arc-based CCS noise, there is no need to add unconditional pin-based CCS noise groups onto the from and to arc pins.

Any output pin that has at least one arc without arc-based CCS noise leading to it must have a pin-based CCS noise.

Furthermore, conditional CCS noise groups can complement the arc-based CCS noise for better accuracy in special cases. Refer to the section on “Liberty Syntax for Standard Cells” for the example on the NAND cell.

Sensitization and Load for Input CCBs

Ideally, the CCS noise is characterized in the context of the entire cell. However, practical safe simplifications can result in faster characterization. For example, only the input CCB that is being characterized and its immediate load (next CCB) are necessary for the characterization.

However, on complex cells the state of other inputs can impact loading of the first-stage CCB, as shown in Figure 8. The recommended approach is to sensitize for the lightest load. For example, A1→N_10 should be characterized with pass gate MM18/MM9 off, which shields the N_10 from capacitance of N_12.

Alternatively, the input conditions that sensitize MM18/MM9 can be enumerated and saved in the library as multiple conditional CCS noise groups. This may result in improvement in accuracy of timing and noise analysis, but the size of the library will increase.

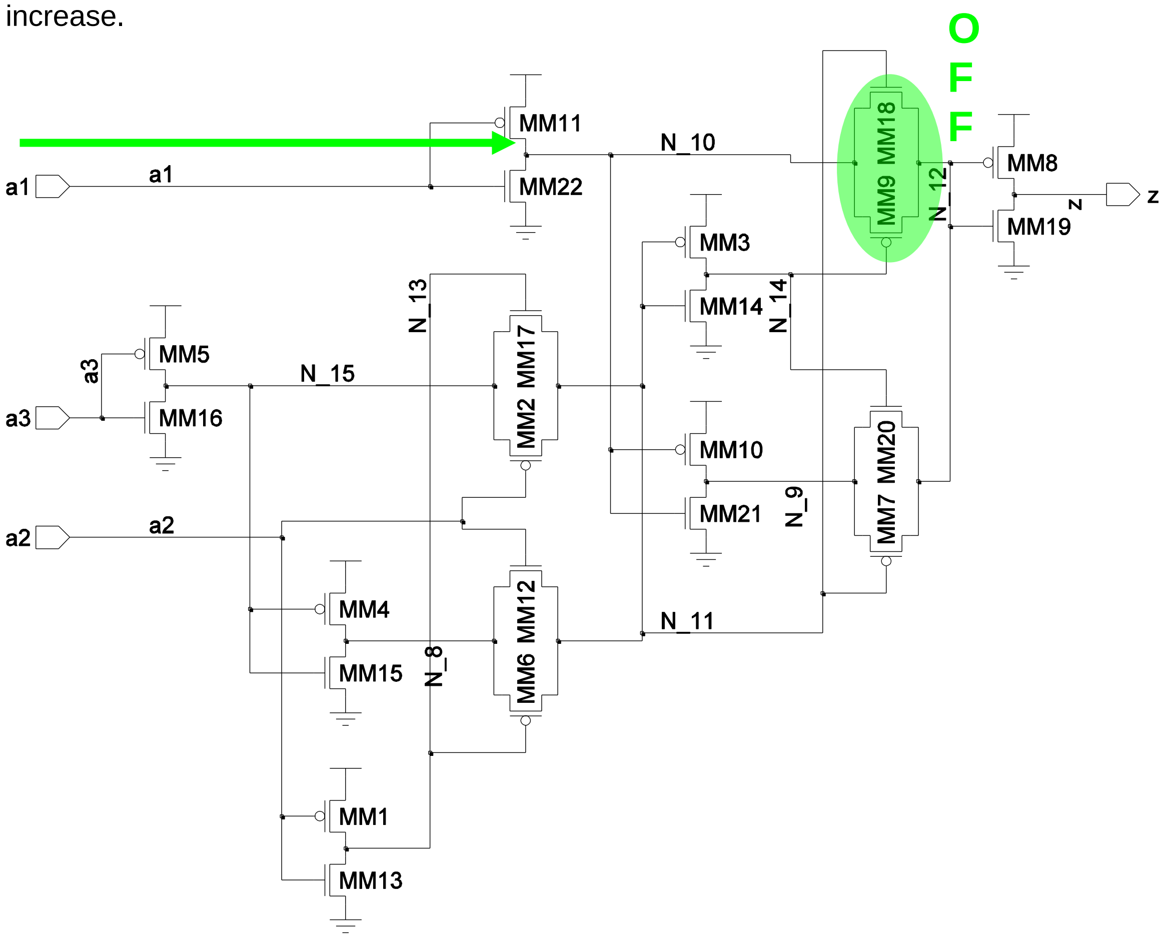


Figure 8 Example of a complex cell where state of other inputs impact the loading on the input CCB

Sensitization and Load for Output CCBs

Similarly to the input CCB, the output CCB ideally is characterized in the context of the entire cell. However, because the input stimuli for CCS noise characterization is generated by the voltage source, the impact of the previous stages is eliminated. Therefore, only the last stage is necessary for the characterization, such as the MM8/MM19 in Figure 8.

Partial Netlist for CCB

Characterization of a partial cell netlist generally results in sufficient accuracy. However, an example of an effect that cannot be captured when using a partial netlist is voltage drop within the cell rail due to current that would be drawn by stages that were cut out. This applies mostly to input CCBs of large cells with detailed parasitics extracted for rail nets.

Another example for output CCBs is the effect of coupling between the last and some other previous CCBs.

Parasitics Handling

Parasitics need to be considered for the following:

- Injection nodes for stimuli
- Observation points
- Any netlist modification, such as partial netlist

The effect of selection of nodes for characterization is minimal for most cells, as long as the parasitics are included. The general recommendation is to include as many parasitics as possible in the characterized partial netlist. For the AND gate shown in Figure 9, the following are example for internal node selection:

- For two separate pin-based CCS noise, use A1→N_7 and N_6→Z. The voltage drop on R7 thus appears in both pin-based CCS noise models.
- For arc-based CCS noise, the intermediate point ideally needs to be same: A1→N_6 and N_6→Z. Alternatively, without any loss of accuracy: A1→N_7 and N_7→Z.
- The coupling capacitors such as Cc3_3 have to be grounded rather than removed for the output CCB that includes MM3/MM6.

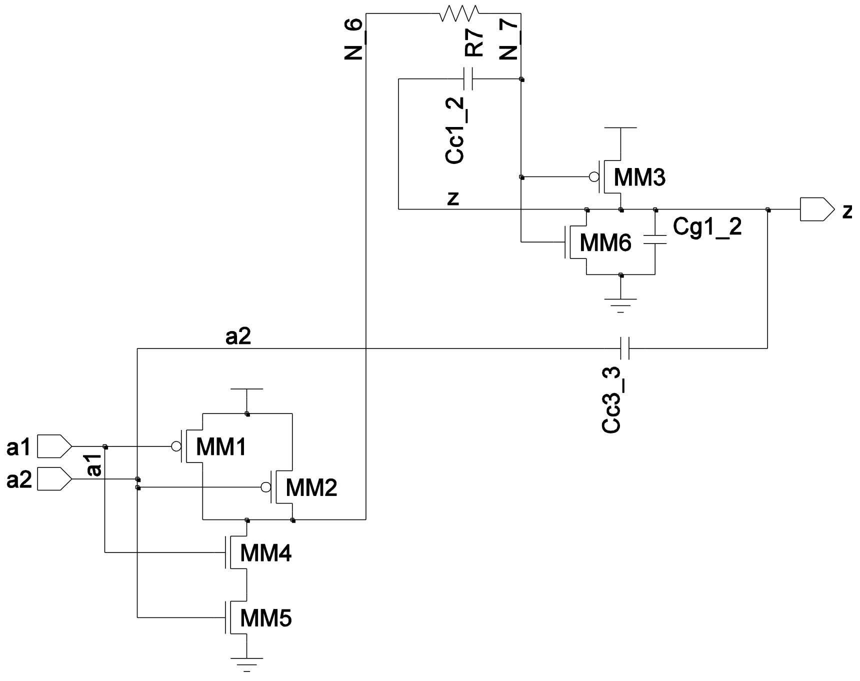


Figure 9 Example of AND Gate with Parasitics

Conditional CCS noise Groups

The CCS noise is a structural boundary model. Therefore, it might not be able to capture all conditional behaviors of the cell, especially on the output stages. Use default CCS noise to capture representative CCS noise properties as follows.

- Input pins: Select the conditions that sensitize the fastest CCS noise. For scaling, select the same to node as the nominal library. The set of CCS noise does not need to be exhaustive but when it is, it reduces pessimism in the analysis tools.
- Input pass gates: The on or off conditions of the input pass gates are fully enumerated.
- Output pins: Select the conditions that sensitize the slowest CCS noise.
- Capturing static output strength not covered by arc-based CCS noise data. For example, when NAND, A=0, B=0, the NAND/Z also has pin-based CCS noise with condition A=0, B=0. The analysis tool can use arc-based CCS noise for accurate propagation or static noise analysis if A=1 or B=1, and the pin-based CCS noise if A=0, B=0.
- Internal node and state in conditions: No internal node or state is allowed in conditions. The selection of CCS noise has to be conservative if the condition cannot be described by input nodes.

  Note that in some cases among many outputs CCS noise groups there can be one CCS noise that is weaker for above low noise and another CCS noise weaker for below high noise. To ensure accurate analysis, include the two worst-case CCS noise groups with their respective when condition. Note that whenever a CCS noise is used as a representative of multiple potential CCS noise groups, the use of the actual condition (when) is strongly recommended to simplify the process of validating the results of static noise analysis tool with SPICE.

  For pin-based CCS noise model, the “when” condition must be specified. In the example below, the “when” condition for CCS noise model on pin “a” must be specified. If the “when” condition is not specified, the arc from a to zn would still be used even when b is set to zero.

```
cell (ND2) {
  pin (zn) {
    timing () {
      related_pin : “a”;
      ccsn_first_stage () {
        ## arc-based CCSN model
        ...
      }
    }
  }
  pin (a) {
    ccsn_first_stage () {
      when : “b”;
      ## pin-based CCSN model
      ...
    }
  ...
}
```

Conservative Selection of Representative CCS noise Groups

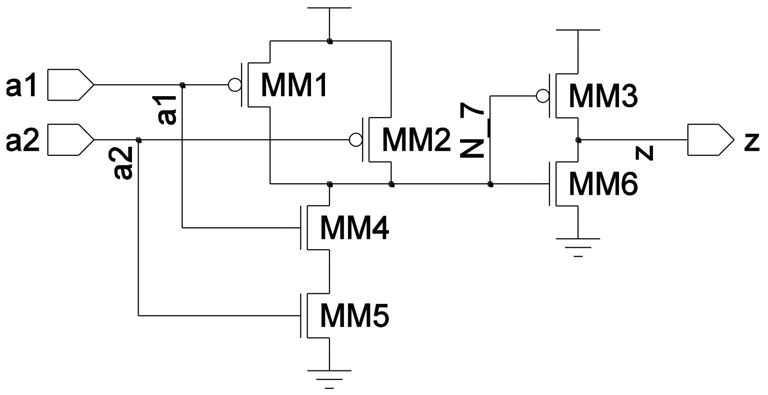
The previous section refers to a representative faster or stronger CCS noise model. Given two characterized CCS noise models, the faster or stronger model is the one that can deliver more current for the same input stimulus or, more accurately, its propagated noise is larger.

The simplest way to estimate the relative strength is to compare sample entries in the CCS noise DC table. More accurately, the timing or propagated noise tables can be compared to account for dynamic effects.

Liberty Syntax for Standard Cells

The following section explains Liberty CCS noise syntax for various cell and library types. In most cases only the CCS noise syntax is shown. Syntax that is not CCS noise specific is shown in regular text, and CCS noise specific syntax is shown in blue bold text.

Multi-Input Two-Stage Cell



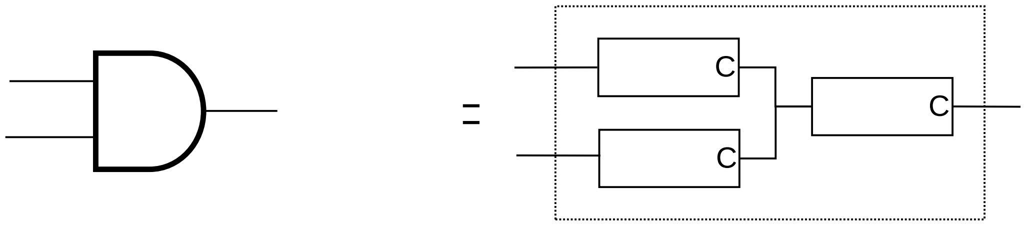


Figure 10 AND Gate

The two-stage cell is described by a set of CCS noise models as follows.

- CCS noise for the first stage. There is one set of data per input, A1→N_7 and A2→N_7.
- CCS noise for the last stage N_7→Z.

The CCS noise for the last stage is listed twice in the Liberty syntax to allow an arc-based description of a propagating path from inputs to outputs.

```
cell(an02d0) {

  pin(A1) {
    direction : input;
    …
  }
  pin(A2) {
    direction : input;
    …
  }
  pin(Z) {
    direction : output;
    …

    timing() {
      related_pin     : "A1";
      ccsn_first_stage() { /* A1 to N_7 */
        …
      }
      ccsn_last_stage() { /* N_7 to Z */
        …
      }
    }
    timing() {
      related_pin     : "A2";
      ccsn_first_stage() { /* A2 to N_7 */
        …
      }
      ccsn_last_stage() { /* N_7 to Z, copy of the above */
        …
      }
    }
  }
}  /* cell(an02d0) */
```

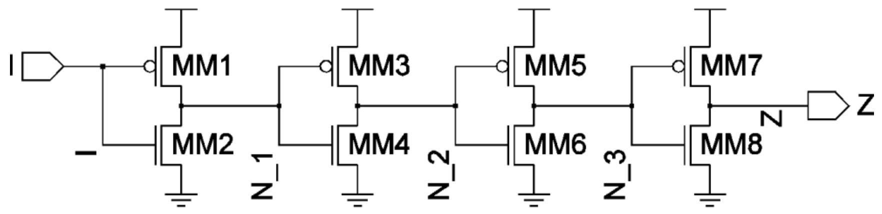
Combinational Cells with Three or More Stages

In general, timing arcs that cannot be modeled with arc-based CCS noise should be modeled as pin-based CCS noise.

Cells with Simple Multiple Stages

Cells with three or more stages should be modeled with pin-based CCS noise. The example below is a delay cell consisting of four inverters in a chain. This cells should be modeled using pin-based CCS noise as followed.

- The ccsn_first_stage at input pin I models the first CCB from I to N_1.
- The ccsn_last_stage at output pin Z models the last CCB from N_3 to Z.



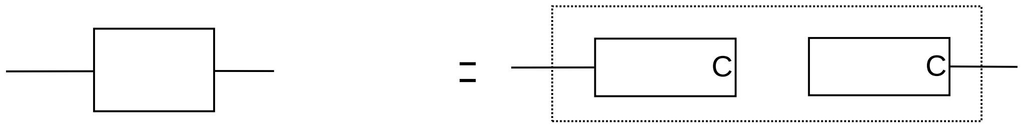


Figure 11 Cells with Simple Multiple Stages

Cells with Multiple Parallel Transistor-Level Paths

It is also recommended to use pin-based CCS noise for timing arcs with two or more active parallel transistor-level paths with different electrical properties. The term active here means nets on the transistor path in question are switching. In the example cell below in Figure 12, for the timing arc from input a1 to output Z, there are two parallel paths:

- Path 1: A1→N_10→N12→Z (2-stage)
- Path 2: A1→N_10→N_9→N_12→Z (3-stage)

Both paths are actively switching when the timing arc a1->Z switches. Therefore, pin-based CCS noise models should be used.

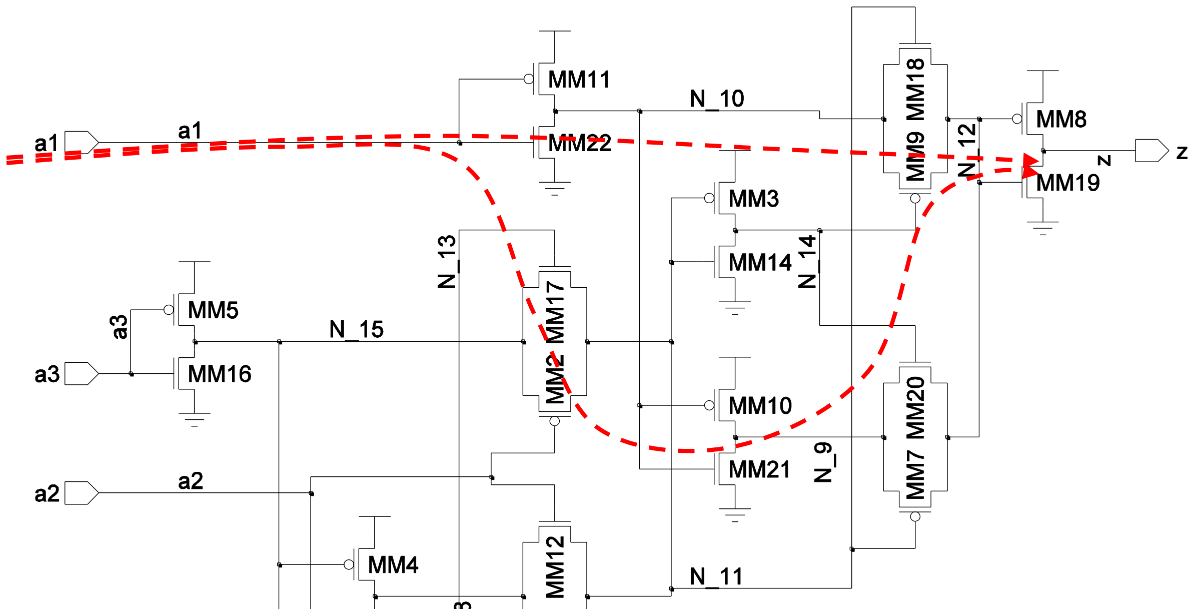


Figure 12 Cell with Multiple Parallel Transistor-Level Paths

Note that the above principle also applies to a timing arc with two active parallel paths one with 1 stage and the other with 2 stages. This type of transistor-level topology occurs more often with signals connected to pass-gate control pins. In the example show in Figure 13, for the timing arc from S to Z, there are 2 active transistor-level paths.

- Path 1 with 2 stages: S->N_1->Z
- Path 2 with 1 stage: S->Z

And both of them are actively switching when the timing arc switches. Therefore, pin-based CCS noise models should be used.

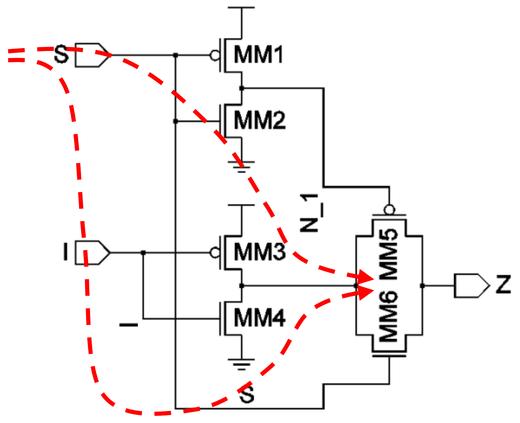
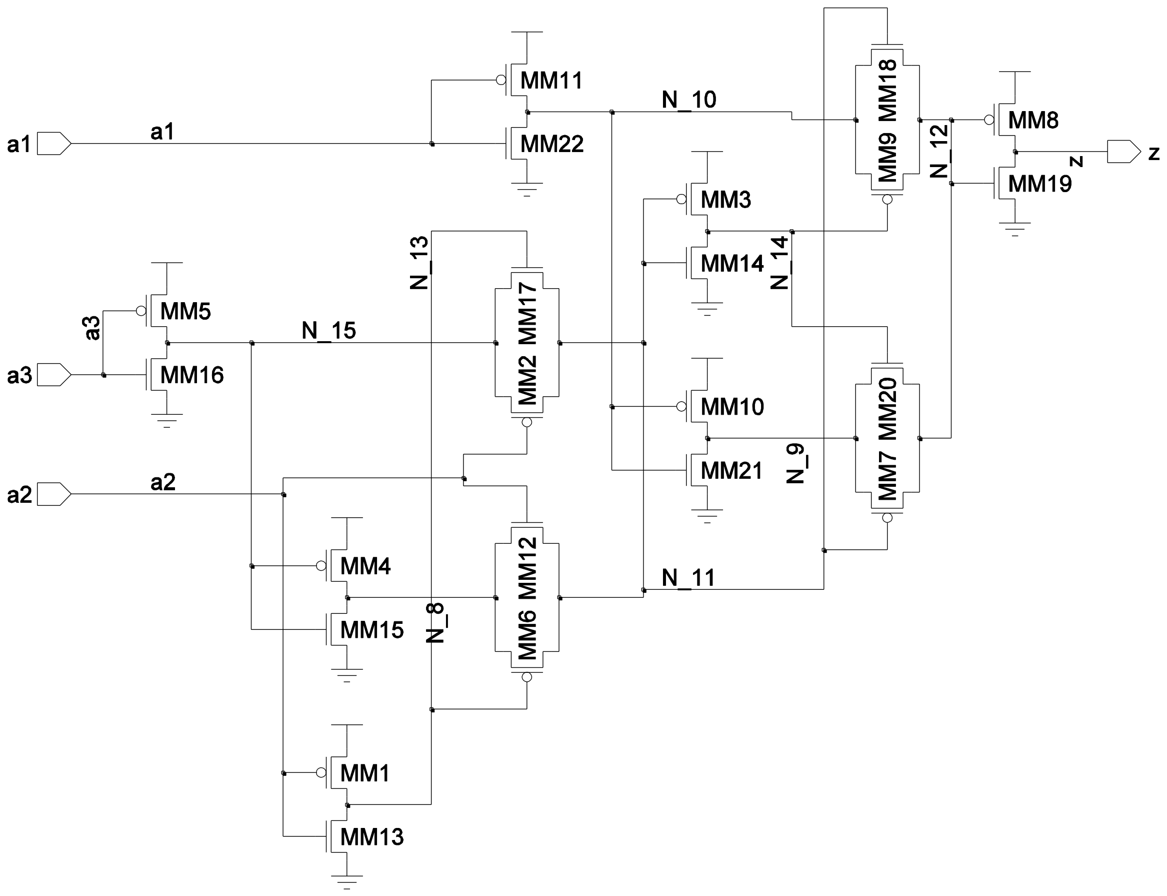


Figure 13 Cell with Multiple Parallel Transistor-Level Paths and Different Number of Stages

3-input XOR Cell Example

The example of pin-based CCS noise model for a 3-input XOR gate is shown below.



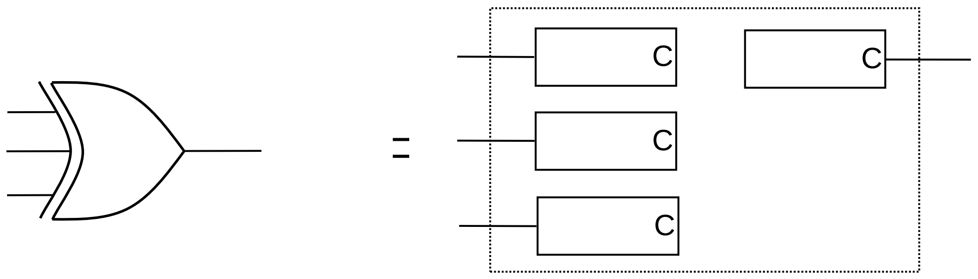
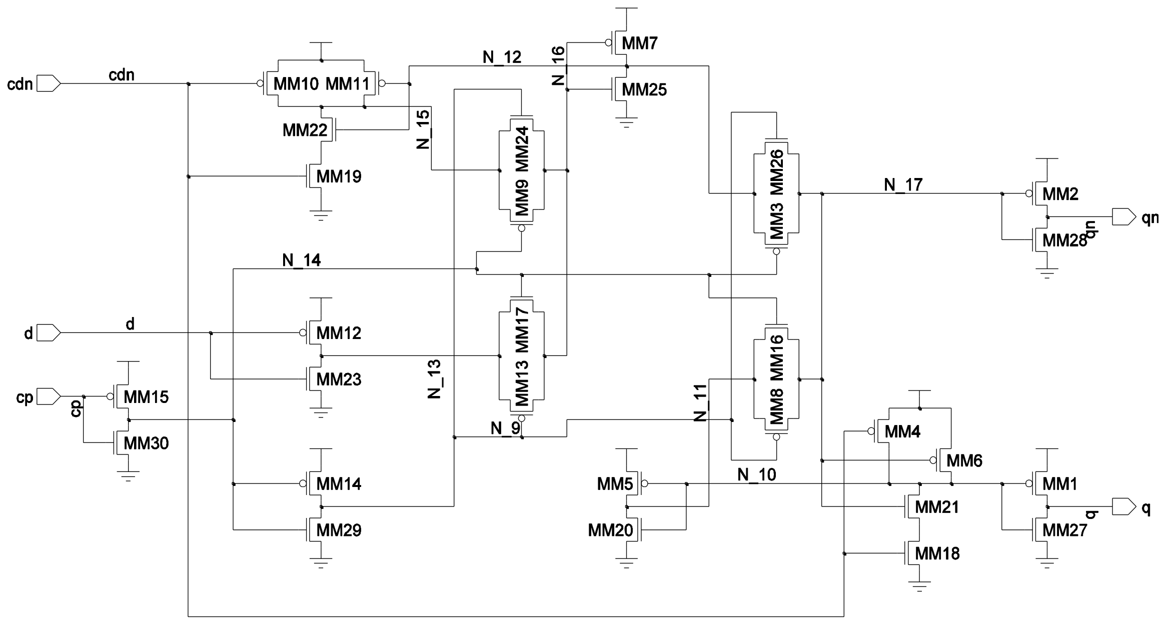


Figure 14 3-Input XOR Example

```
cell(xr03d1) {
  …
   Function : “A1^A2^A3”;

  pin(A1) {
    direction : input;
    …
    ccsn_first_stage() { /* A1 to N_10 */
      …
    }
  }

  pin(A2) {
    direction : input;
    …
    ccsn_first_stage() { /* A2 to N_8 */
      …
    }
  }

  pin(A3) {
    direction : input;
    …
    ccsn_first_stage() { /* A3 to N_15 */
      …
    }
  }

  pin(Z) {
    direction : output;
    …

    ccsn_last_stage() { /* N_12 to Z */
      …
    }
  }
}  /* cell(xr03d1) */
```

Edge Triggered Flip-Flop



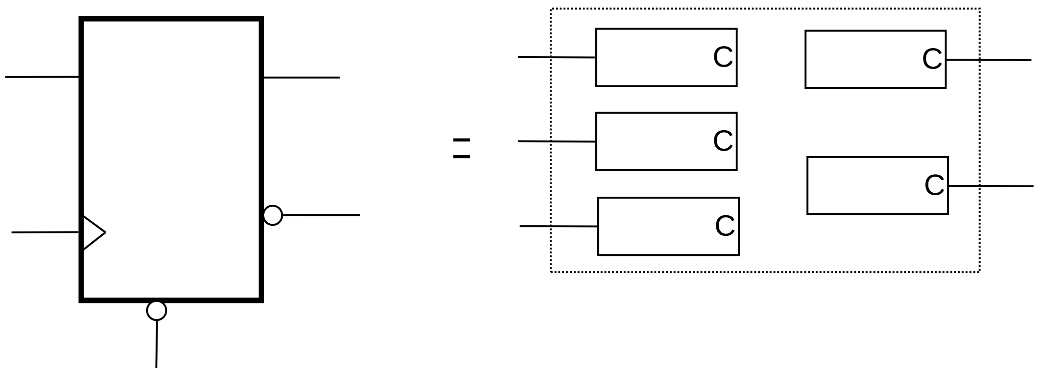


Figure 15 Edge Triggered Flip-Flop

A boundary model of the flip-flop consists of CCS noise for the first stage driven by the inputs and the last stage driving the outputs.

If there are paths of up to two stages from inputs to outputs, they can be represented by arc-based CCS noise (timing group). In this example there are no such paths, therefore all CCS noise groups are on pins (pin group).

```
cell(dfcrb1) {

  pin_opposite("QN","Q");
  ff (IQ,IQN) {
    clocked_on : "CP" ;
    next_state : "D" ;
    clear : "CDN'" ;
  }
  pin(D) {

    direction : input;
    timing() {
      timing_type     : setup_rising;
      related_pin     : "CP";
    }
    timing() {
      timing_type     : hold_rising;
      related_pin     : "CP";
    }
    ccsn_first_stage () {  /* from D to N_13 */
      ...
    }
    /* In addition to the above the D to N_16 can be described
     * by a conditional CCS noise for best accuracy. Note that in this
     * particular case it is not necessary since very likely
     * the N_13 to N_16 attenuates noise only minimally.
     * Another alternative description of the cell is to output
     * unconditional
     * CCS noise D to N_16 since there is no fanout on N_13.
     *
     * ccsn_first_stage () {  /* from D to N_16 */
     *   ...
     *   when : "!CP";
     *   ...
     * }
     */
  }

  pin(CP) {
    direction : input;
    clock : true;
    ccsn_first_stage () {  /* from CP to N_14 */
      ...
    }
  }

  pin(CDN) {
    direction : input;
    timing() {
      timing_type     : recovery_rising;
      related_pin     : "CP";
    }
    timing() {
      timing_type     : removal_rising;
      related_pin     : "CP";
    }

    /* Pin CDN has a fanout of 2 with potentially reconverging paths. A
     * complex way to most accurately describe it is to analyze all
     * paths from CDN and determine whether all terminate at the output
     * pins and whether they reconverge. If any one path does not reach
     * the output or they reconverge at a node other than the CCS noise
     * stage output then pin-based CCS noise must be
     * used. A simple way to represent the cell without much
     * analysis is to use pin-based CCS noise if fanout of 2
     * or more is encountered on the pin, which is shown below:
     */
    ccsn_first_stage () {  /* from CDN to N_15, see the figure below */
      ...
      when : "!CP&D";
      ...
    }
    ccsn_first_stage () {  /* from CDN to N_10 */
      ...
      when : "IQ";
      ...
    }
    /* A simpler alternative to having 2 conditional CCS noise
     * groups is only
     * use one of them: select the one that propagates more noise
     * (stronger/faster). The selected single CCS noise may or may
     * not contain the condition (when).
     */
  }

  pin(Q) {
    direction : output;
                function   : "IQ";
    timing() {
      timing_type      : rising_edge;
      related_pin      : "CP";
    }
    timing() {
      timing_type      : clear;
      timing_sense     : positive_unate;
      related_pin      : "CDN";
    }
    ccsn_last_stage () {   /* from N_10 to Q */
      ...
    }
  }

  pin(QN) {
    direction : output;
                function   : "IQN";
    timing() {
      timing_type      : rising_edge;
      related_pin      : "CP";
    }
    timing() {
      timing_type      : preset;
      timing_sense     : negative_unate;
      related_pin      : "CDN";
    }
    ccsn_last_stage () {   /* from N_17 to QN */
      ...
    }
  }
}  /* cell(dfcrb1) */
```

The CDN to N_15 condition (when statement) is shown below in Figure 16. Red color indicates logic 1, and green color indicates logic 0.

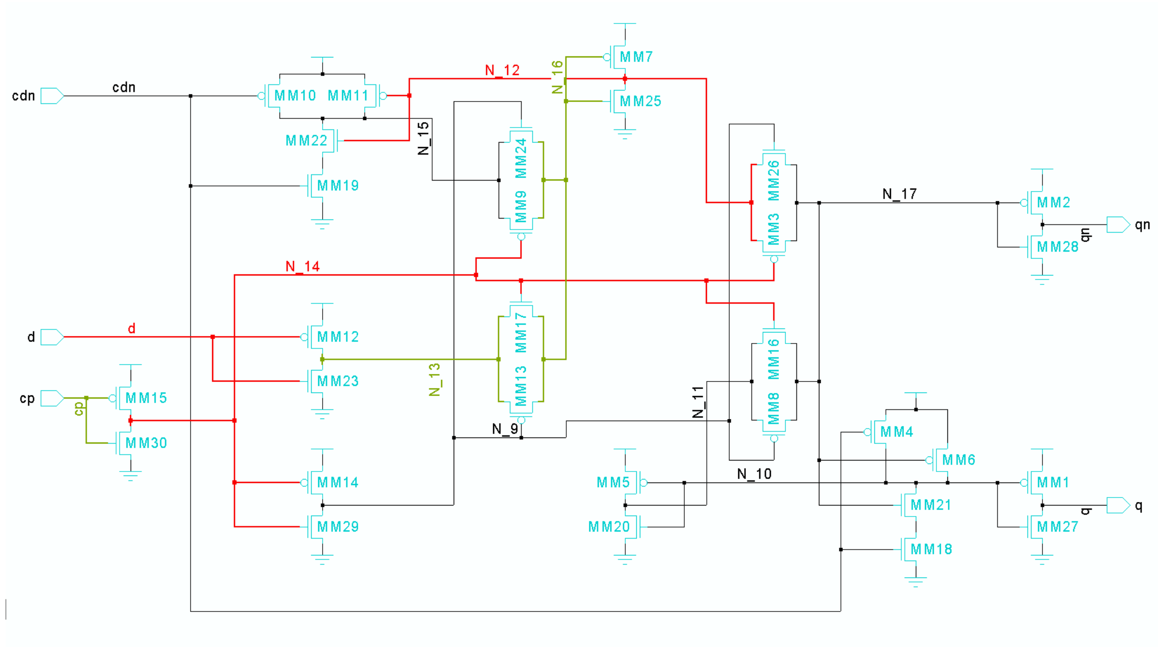


Figure 16 Edge Triggered Flip-Flop for Conditional Statement

Tie-Off Cell

A tie-off cell does not have input pins, therefore only a small portion of CCS noise model syntax is allowed on tie-off cells. The tie-off behavior of a pin is indicated by existing syntax “function” or “driver_type”.

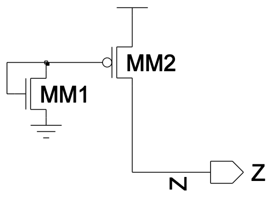


Figure 17 Tie-Off Cell

```
cell (TIEHIGH) {
  dont_touch : true;
  pin(Z) {
    direction : output;
    driver_type : pull_up;
    function :  "1";
    ...
    ccsn_last_stage () {
      /* !!! Only static info allowed */
      stage_type : pull_up;
      dc_current(...) {...}
      /* The following is not allowed
       *   ccsn_timing_*
       *   propagated_noise_*
       *   miller_cap_*
       *   is_inverting
       */
      /* The when is allowed and optional. It needs to be defined on
    * pins of complex cells if the pin gains the tie-off
    * functionality conditionally.
```

```
         */
      }
    }
  }
```

Load Cell

Load cells do not need a detailed CCS noise model. Therefore, the flag is_needed is set to false. This is necessary to inform the analysis tool that this cell is not impacted by noise and does not provide any active current source.

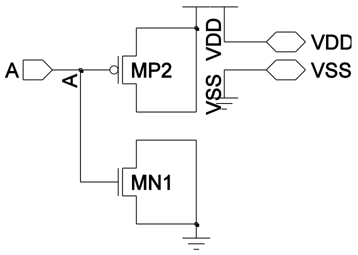
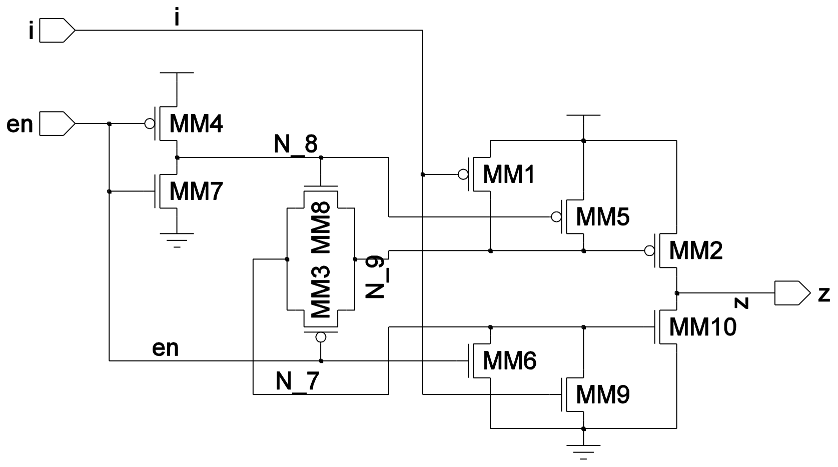


Figure 18 Load Cell

```
cell(LOAD) {
  pin(A) {
    direction : input;
    ccsn_first_stage () {
      is_needed : false;
    }
  }
} /* end LOAD */
```

Three-State Cells

Although the three-state cell shown in Figure 19 has paths with 2 stages from I to Z and EN to Z, it is modeled by pin-based CCS noise groups. This is necessary because there are multiple reconvergent paths such as EN→N_8→Z (no pass gate) and EN→N_8→N_7→Z (through the control of the pass gate). The CCS noise model cannot directly benefit from the arc-based propagating reconvergent paths. Therefore, simpler pin-based representation is sufficient. However, when EN=0 then I→Z becomes a single path which thus can be described by conditional CCS noise.



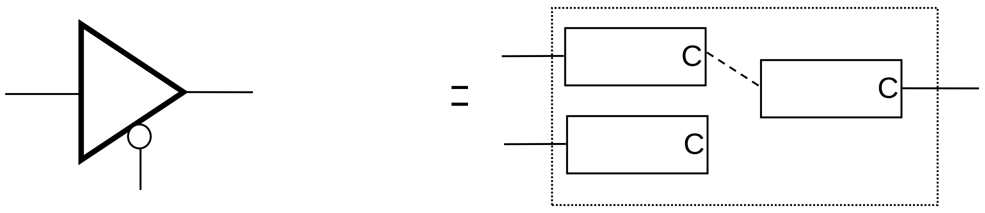


Figure 19 Three-State Cells

```
cell(buftd1) {

  pin(EN) {
    direction : input;
    ccsn_first_stage () {  /* from EN to N_8 */
      ...
    }
    /* Optionally for slightly better accuracy
     * the path EN to N_7 or N_8 can be described too
     * ccsn_first_stage () {  /* from EN to N_7 */
     *    ...
     *    when : "!EN";
     *    ...
     * }
     */
    }
  }

  pin(I) {
    direction : input;
    ccsn_first_stage () {  /* from I to N_7 */
      ...
      when : "!EN"; /* To make sure the pass gate MM3/MM8 is ON */
      ...
    }
  }

  pin(Z) {
    direction : output;
    function  : "I";
    three_state : "EN";
    timing() {
      timing_type     : three_state_enable;
      related_pin     : "EN";
      timing_sense    : non_unate;
      ...
    }
    timing() {
      timing_type     : three_state_disable;
      related_pin     : "EN";
      timing_sense    : non_unate;
      ...
    }
    timing() {
      related_pin     : "I";
      when            : “!EN”;
      timing_sense    : positive_unate;
      ...
   /* Since the I to Z path is a single 2-stage path when EN=0
    * then the I to Z arc can contain conditional CCS noise
    *   ccsn_first_stage () {…}
    *   ccsn_last_stage () {…}
    * instead of the CCS noise on pins Z and I.
    * Unlike NLD-Timing today the I->Z arc must contain “when”.
    */

    }
    ccsn_last_stage () {   /* from N_7 to Z, enabled */
      ...
      when : "!EN";
      ...
    }
    ccsn_last_stage () {   /* from N_7 to Z, disabled */
      when : "EN";
      is_needed : false;
    }
  }

}  /* cell(buftd1) */
```

Three-State Cells With Independent Output Control Paths

Another type of three-state cell does not have full inverter structure on the output but rather independently controlled P and N transistors. In the example shown in Figure 20, N_UP controls MM7, and N_DOWN controls MM6. The CCS noise data could be modeled by two half-unate CCS noise groups, N_UP →Z and N_DOWN→Z, if these paths were highly asymmetrical in delay. Because most cell designs balance the paths, the simplest yet accurate model is to characterize CCS noise for the inverter consisting of MM7 and MM6.

In addition, use pin-based CCS noise inputs because there are reconvergent paths, for example, I→N_UP→Z and I-→N_DOWN→Z.

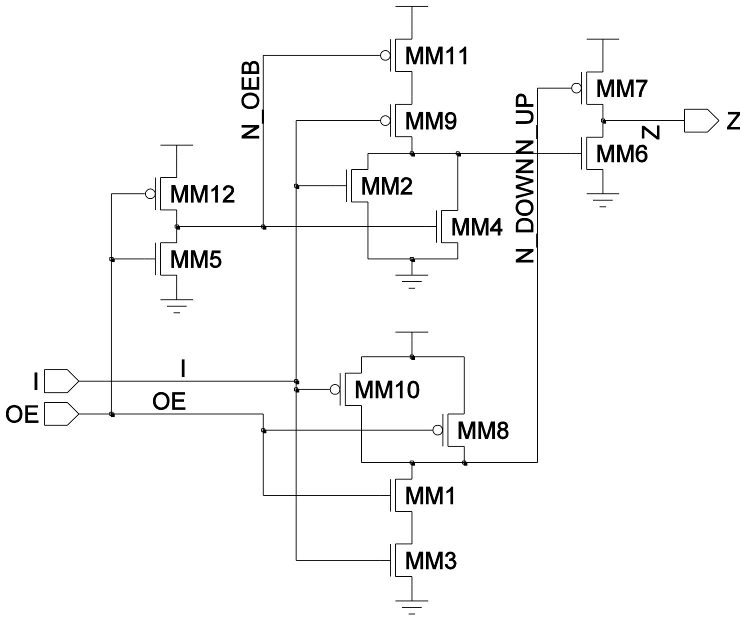


Figure 20 Three-State Cells With Independent Output Control Paths

```
cell(buf_indep) {

  pin(OE) {
    direction : input;
    ccsn_first_stage () {  /* from OE to N_OE */
      ...
    }
    /* Optionally for slightly better accuracy
     * the path OE to N_UP can be described too
     * ccsn_first_stage () {  /* from OE to N_UP */
     *    ...
     *    when : "I";
     *    ...
     * }
     */
    }
  }

  pin(I) {
    direction : input;
    ccsn_first_stage () {  /* from I to N_UP */
      ...
      when : "OE";
      ...
    }
    ccsn_first_stage () {  /* from I to N_DOWN */
      ...
      when : "!OE";
      ...
    }
  }

  pin(Z) {
    direction : output;
    function  : "I";
    ccsn_last_stage () {  /* from shorted N_UP, N_DOWN to Z */
      ...
      when : "OE";
      ...
    }
    ccsn_last_stage () {  /* disabled */
      when : "!OE";
      is_needed : false;
    }
  }

}  /* cell(buf_indep) */
```

Multi-rail Cells

Multi-rail cells are characterized similarly to any multi-stage cell. The input CCB and separately output CCB are typically powered by single voltage. All measurements refer to that voltage, which might be different for each CCB.

Define the rails and their association with using an existing syntax as defined by the Library Compiler Reference Manual, such as power_supply, input/output_signal_level.

Half-Unate (Pull-Up/Pull-Down) Cells

Cell pins sensitive to only high or low signal level are modeled with partial CCS noise model. Such pins are typically found on a single (sleep) transistor cells or multiplexer-like structures. A sample partial structure (not a complete cell) is shown in Figure 21.

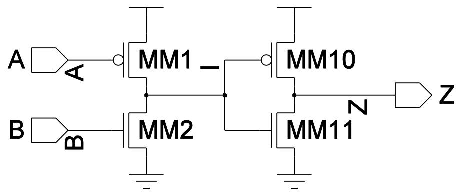


Figure 21 Half-Unate Cells

```
cell(half_sample) {

  pin(A) {
    direction : input;
    ccsn_first_stage () {  /* from A through MM1 */
      /* when : "!B"; optional */
      /* Only pull-up CCS noise */
      stage_type : "pull_up";
      is_inverting : true;
      miller_cap_rise : ...;
      dc_current(...) {...}
      output_voltage_rise(...) {...)
      propagated_noise_low(...) {...}
    }
  }

  pin(B) {
    direction : input;
    ccsn_first_stage () {  /* from B through MM2 */
      /* when : "A"; optional */
      /* Only pull-down CCS noise */
      stage_type : "pull_down";
      is_inverting : true;
      miller_cap_fall : ...;
      dc_current(...) {...}
      output_voltage_fall(...) {...)
      propagated_noise_high(...) {...}
    }
  }

  …
}
```

Pass Gate Modeling

For pass gate modeling, it is recommended to create a CCS noise model with the entire pass gate included in the CCB, and a CCS noise model for the inverter by itself. In the figure below, it is recommended not to have separate CCS noise models for the pullup and pulldown transistors.

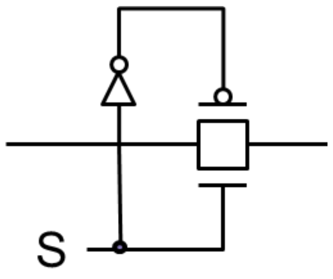


Figure 22 Pass Gate Modeling

Unbuffered Combinational Cells

The input (or output) pass gate is simply added as a part of the first (last) CCB as shown on the following partial cell structure.

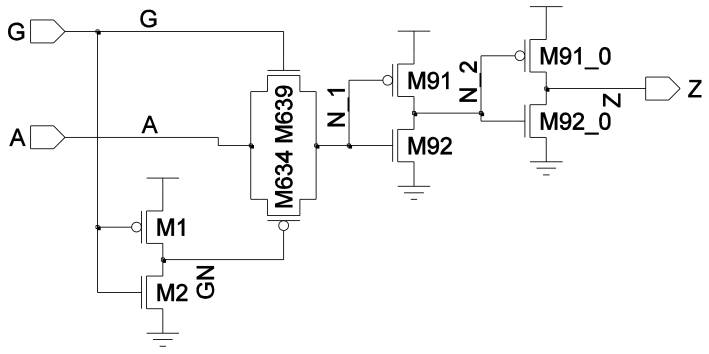


Figure 23 Unbuffered Combinational Cells

```
cell(i_pass) {

  pin(A) {
    direction : input;
    …
    }
  }

  pin(G) {
    direction : input;
    …
    }
    ccsn_first_stage () {  /* from G to GN */
      ...
    }
    /* Optionally a two half-unate CCS noise groups can be added
     * for M639 G->N_1
     * if the M639 is much stronger (faster, lower Vth) than M2
     *   ccsn_first_stage () {  /* from A to N_2 */
     *     /* pull-down only */
     *     when : "!A";
     *     ...
     *   }
     *   Also, the weak pull-up can be represented, but generally
     *   not needed since it is weak.
     *   ccsn_first_stage () {  /* from A to N_2 */
     *     /* weak pull-up only */
     *     when : "A";
     *     ...
     *   }
     */

  }

  pin(Z) {
    direction : input;
    …
  }
  pin(Z) {
    direction : output;
    timing() {
      related_pin      : "I";
      timing_sense     : positive_unate;
      …
      ccsn_first_stage () {  /* from A to N_2 */
        . . .
        when : "G";
        . . .
      }
      ccsn_last_stage () {  /* from N_2 to Z */
        …
      }
  }

}  /* cell(i_pass) */
```

Bidirectional Cells

Bidirectional pins are described by one CCS noise for the input stage and one for the output stage. Alternatively, arc-based description can be used as long as it is proper when added such that the driver is disabled for the input CCS noise or enabled for the output CCS noise.

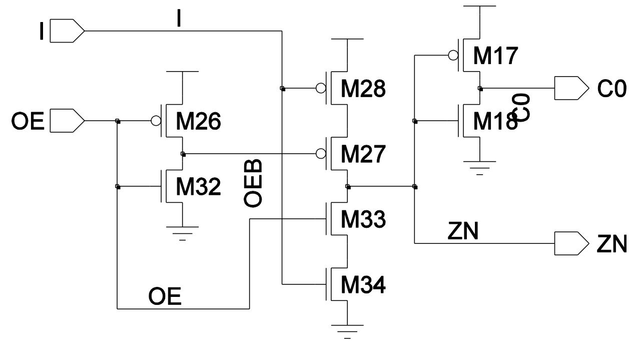


Figure 24 Bidirectional Cells

```
cell(bid) {

  pin(OE) {
    …
    ccsn_first_stage () {  /* from OE to OEB */
      . . .
    }
  }

  pin(I) {
    …
  }

  pin(C0) {
    …
    timing() {
      related_pin      : "ZN";
      timing_sense     : negative_unate;
      when             : "!OE";
      . . .
      ccsn_first_stage () {  /* from ZN to C0 */
```

```
        ...
      }
    }

    pin(ZN) {
      direction : inout;
      function  : "I’";
      three_state : "OE’";
      timing() {
        related_pin     : "I";
        timing_sense    : negative_unate;
        when            : “OE”;
        ...
        ccsn_first_stage () {  /* from I to ZN */
          ...
        }
      }
    }

  }  /* cell(bid) */
```

Library Syntax for Macro Blocks

Because CCS noise is a boundary model, any macro block can be viewed as a large cell. The only major syntactical difference is that CCS noise can be defined on buses, both pin-based or arc based.

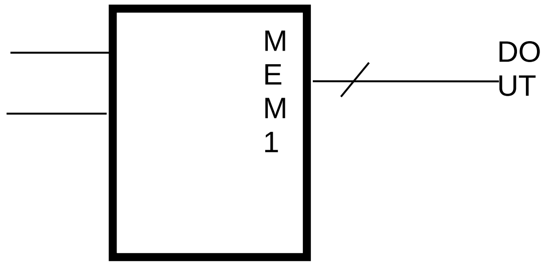
Arc- and Pin-based CCS noise

In most cases macros contain many complex paths. Therefore, a majority of CCS noise models will be pin-based. The same rules apply as for standard cells. If a path is of length two or less, it can be represented by arc-based CCS noise.

CCS noise on Buses

Similarly to NLD-timing tables, the CCS noise can be represented by one entry per bus. The selection of which bus pin is characterized as a representative of a bus should match the selection of the pin for NLD-timing. This is sufficient due to high symmetry of bus design and layout (shown in the following example).

Alternatively, CCBs of all individual bus pins can be characterized and represented in the library similarly to NLD-timing. A simplification that might result in slight loss of accuracy for such asymmetric busses is to use bus-based CCS noise syntax by selecting the strongest CCS noise for inputs, and the weakest one for outputs.



```
type (BUS_128) {
  base_type : array;
  data_type : bit;
  bit_width : 128;
  bit_from : 127;
  bit_to : 0;
  downto : true;
}

cell (mem1) {
  ...
  memory() {
    type : ram;
    address_width : 8;
    word_width : 128;
  }

  bus (DATAOUT) {
    bus_type : "BUS_128";
    direction : output;
    ccsn_last_stage () {
      …
    }
    timing() {
      timing_sense : non_unate;
      ...
    }
  }
  ...
}
```

CCS Noise Liberty Extension (June 2016)

The CCS noise syntax was originally developed for accurate computation of crosstalk noise and crosstalk delay analysis. With advanced waveform propagation, PrimeTime uses both CCS timing and CCS noise library data in accurate waveform-based timing analysis. Liberty CCS noise syntax was extended to improve the precision and accuracy of CCS noise library modeling for timing analysis purpose.

This section describes the key changes to CCS noise Liberty syntax.

Channel-Connected Block Representation

The `ccsn_first_stage` and `ccsn_last_stage` blocks are replaced by `input_ccb` and `output_ccb`, respectively. These blocks contain CCS noise measurement data from characterization. There is no change to `miller_cap_rise` or `max_cap_fall` or noise propagation tables.

Each `input_ccb` and `output_ccb` block is required to have a name string, which must be unique within a pin group. The name string is used to identify which CCS noise model will be referenced from the `timing` group or `receiver_capacitance` group. Since the name string is used, this avoids multiple definitions of the same channel-connect block in different `timing` or `receiver_capacitance` groups. This is shown in *Example 8* for the 2-input AND gate, where `output_ccb` named *ccb3* is referenced from both `timing` groups for "A1" and "A2" pins.

Each timing group contains subset of the following new attributes, depending on the timing arc type: `propagating_ccb`, `active_input_ccb`, `active_output_ccb`.

- Propagating timing arc: `propagating_ccb`
- Non-propagating timing arc: `active_input_ccb` and `active_output_ccb`

Pin-based receiver_capacitance group

- Switching receiver model is recommended to have `active_input_ccb`.
- Non-switching receiver model should not contain `active_input_ccb`.

Characterization Waveform & Voltage Measurement

For CCS noise characterization using Liberty syntax prior to this extension, it is recommended to use saturated ramp waveform, and the transition time was measured from rail to rail. With this Liberty syntax extension, it is recommended to use normalized driver waveform, and the transition time is measured between slew trip points to be consistent with NLDM and CCS timing data. All other library attributes that pertain to slew attributes, such as `slew_derate_from_library`, are consistently applied to NLDM, CCS timing, and CCS noise library data.

For each vector or simulation, it is still recommended to use 5 time and voltage pairs. With Liberty syntax prior to this extension, it is recommended to measure the output voltage waveform when it crosses (0.10, 0.30, 0.50, 0.70, 0.90) of VDD. With Liberty syntax extension, the output voltage waveform measurement should include the slew trip points. The first and last measurements should also be adjusted for the slew trip points. The output voltage waveform should be measured when it crosses the following points: (slew_lower_threshold/2, slew_lower_threshold, 0.50, slew_upper_threshold, (1 + slew_upper_threshold)/2) of VDD.

- If the slew trip points are at 30% and 70% of VDD, there is no change to the voltage measurement points.
  - Liberty syntax prior to extension: (0.10, 0.30, 0.50, 0.70, 0.90) of VDD
  - Liberty syntax extension: (0.10, 0.30, 0.50, 0.70, 0.90) of VDD
- If slew trip points are at 25% and 75% of VDD, the voltage measurement points need to be updated.
  - Liberty syntax prior to extension: (0.10, 0.30, 0.50, 0.70, 0.90) of VDD
  - Liberty syntax extension: (0.125, 0.25, 0.50, 0.75, 0.875) of VDD

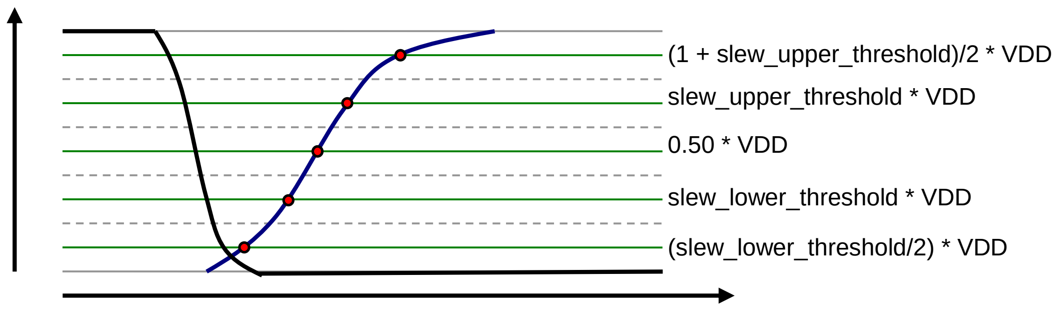

(1 + slew_upper_threshold)/2 * VDD
slew_upper_threshold * VDD
0.50 * VDD
slew_lower_threshold * VDD
(slew_lower_threshold/2) * VDD

Figure 255 Measuring output voltage waveform for CCS noise Liberty extension

Indexes

With Liberty syntax extension, the selection of input slew and load capacitance indexes for CCS noise should be consistent with NLDM and CCS timing tables. To miminize library size, one method is to deploy a table for CCS noise that is 2 times sparser than NLDM and CCS timing tables.

NLDM and CCS timing uses the following 7x7 table:

{s1, s2, s3, s4, s5, s6, s7} x {c1, c2, c3, c4, c5, c6, c7}

The CCS noise table can be reduded to the following 4x4 table:

{s1, s3, s5, s7} x {c1, c3, c5, c7}

Example 4 Reduced CCS noise table for NLDM and CCS timing table with 7x7 indexes

NLDM and CCS timing uses the following 10x8 table:

{s1, s2, s3, s4, s5, s6, s7, s8, s9, s10} x {c1, c2, c3, c4, c5, c6, c7, c8}

The CCS noise table can be reduced to the following 5x4 table:

{s1, s3, s5, s7, s10} x {c1, c3, c5, c8}

Example 5 Reduced CCS noise table for NLDM and CCS timing table with 10x8 indexes

When the CCB output node is internal, Nx1 table is used. For example, if the NLDM & CCS timing table for a buffer cell is 10x8,

- `input_ccb` table should be 10x1
- `output_ccb` table should be 10x8

Examples of CCS noise Liberty Extension

Example 6 Inverter

Without CCS noise Liberty Extension

```
cell (INVD1) {
  pin (I) {}
  pin (ZN) {
    timing () {
      related_pin : "I";
      timing_sense : negative_unate;
      timing_type : combinational;
      ccsn_first_stages () {}
    }
  }
}
```

With CCS noise Liberty Extension

```
cell (INVD1) {
  pin (I) {
     input_ccb (“ccb1”) {}
  }
  pin (ZN) {
    timing () {
      related_pin : "I";
      timing_sense : negative_unate;
      timing_type : combinational;
      propagating_ccb ("ccb1");
    }
  }
}
```

Example 7 Buffer

Without CCS noise Liberty Extension

```
cell (BUFFD1) {
  pin (I) {}
  pin (Z) {
    timing () {
      related_pin : "I";
      timing_sense : positive_unate;
      timing_type : combinational;
      ccsn_first_stages () {}
      ccsn_last_stages () {}
    }
}
```

With CCS noise Liberty Extension

```
cell (BUFFD1) {
  pin (I) {
    input_ccb (“ccb1”) {}
  }
  pin (Z) {
    output_ccb (“ccb2”) {}
    timing () {
      related_pin : "I";
      timing_sense : positive_unate;
      timing_type : combinational;
      propagating_ccb (“ccb1”, “ccb2”);
```

	```
	        }

	    }

	}
	```

Example 8 2-input AND Gate

Without CCS noise Liberty Extension

```
cell (AN2D1) {
  pin (A1) {}
  pin (A2) {}
  pin (Z) {
    timing () {
      related_pin : "A1";
      timing_sense : positive_unate;
      timing_type : combinational;
      ccsn_first_stage () {}
      ccsn_last_stage () {}
    }
    timing () {
      related_pin : "A2";
      timing_sense : positive_unate;
      timing_type : combinational;
      ccsn_first_stage () {}
      ccsn_last_stage () {}
    }
  }
}
```

With CCS noise Liberty Extension

```
cell (AN2D1) {
  pin (A1) {
    input_ccb (“ccb1”) {}
  }
  pin (A2) {
    input_ccb (“ccb2”) {}
  }
  pin (Z) {
    output_ccb (“ccb3”) {}
    timing () {
      related_pin : "A1";
      timing_sense : positive_unate;
      timing_type : combinational;
      propagating_ccb (“ccb1”, “ccb3”);
    }
    timing () {
      related_pin : "A2";
      timing_sense : positive_unate;
      timing_type : combinational;
      propagating_ccb (“ccb2”, “ccb3”);
    }
  }
}
```

Example 9 D Flip-flop

Without CCS noise Liberty Extension

```
cell (DFFD1) {
  pin (CP) {
    ccsn_first_stage () {}
    ccsn_first_stage () {}
    receiver_capacitance () {}
    receiver_capacitance () {}
  }
  pin (D) {
    ccsn_first_stage () {}
    ccsn_first_stage () {}
    receiver_capacitance () {}
    receiver_capacitance () {}
  }
  pin (Q) {
    timing () {
      related_pin : "CP";
      timing_sense : non_unate;
      timing_type : rising_edge;
    }
    timing () {
      related_pin : "CP";
      timing_sense : non_unate;
      timing_type : rising_edge;
    }
    ccsn_last_stage () {}
    ccsn_last_stage () {}
  }
}
```

With CCS noise Liberty Extension

```
cell (DFFD1) {
  pin (CP) {
    input_ccb (“ccb1”) {}
    input_ccb (“ccb2”) {}
    receiver_capacitance () {
      active_input_ccb(“ccb1”);
    }
    receiver_capacitance () {
      active_input_ccb(“ccb2”);
    }
  }
  pin (D) {
    input_ccb (“ccb3”) {}
    input_ccb (“ccb4”) {}
    receiver_capacitance () {
      active_input_ccb(“ccb3”);
    }
    receiver_capacitance () {
      active_input_ccb(“ccb4”);
    }
  }
  pin (Q) {
    output_ccb (“ccb5”) {}
    output_ccb (“ccb6”) {}
    timing () {
      related_pin : "CP";
      timing_sense : non_unate;
      timing_type : rising_edge;
      active_input_ccb (“ccb1”);
      active_output_ccb (“ccb5”);
    }
    timing () {
      related_pin : "CP";
      timing_sense : non_unate;
      timing_type : rising_edge;
      active_input_ccb (“ccb2”);
      active_output_ccb (“ccb6”);
    }
  }
}
```

Summary

This document provides techniques to create accurate CCS noise libraries. Many aspects of characterization are described, including how to partition the cell netlist and select internal nodes for voltage measurements. Selection of pin-based and arc-based CCS noise data is discussed with examples of different cell types, including multi-stage cells. Following the general guidelines in the document will help you to make the appropriate choices for characterization.