

# iEDA Tutorial Agenda

- **Part 0:** iEDA Overview (**Xingquan Li**)
- **Part 1:** iEDA Infrastructure (**Zengrong Huang**)
- **Part 2:** iPL: Placement Tool and Its Technology (**Shijian Chen**)
- **Part 3:** iCTS: Clock Tree Synthesis Tool and Its Technologies (**Weiguo Li**)
- **Part 4:** iRT: Routing Tool and Its Technologies (**Zhisheng Zeng**)
- **Part 5:** iSTA: Static Timing Analysis Tool and Its Technologies (**He Liu**)
- **Part 6:** iPA: Power Analysis Tool and Its Technologies (**Simin Tao**)

# iSTA: An Open-source Static Timing Analysis EDA Tool

Simin Tao<sup>1</sup>, He Liu<sup>2</sup>, Shuaiying Long<sup>1</sup>, Xinyu Ye<sup>3</sup> Yingbin Cheng<sup>4</sup>, Xingquan Li<sup>1</sup>

<sup>1</sup>Peng Cheng Laboratory;

<sup>2</sup>Peking University;

<sup>3</sup>Beijing Institute of Open Source Chip;

<sup>4</sup>Shanghai Tech University;

Email: taosm@pcl.ac.cn, liuh@stu.pku.edu.cn, lixq01@pcl.ac.cn

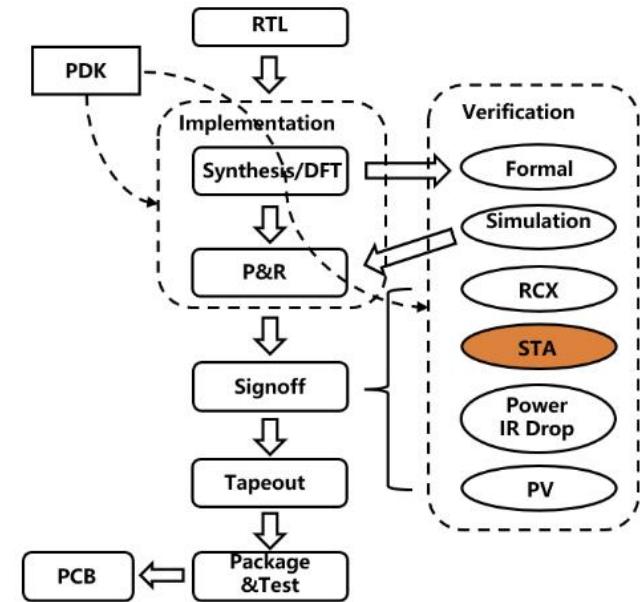


# Content

- 01 Background**
- 02 iSTA structure**
- 03 Timing calculation**
- 04 Timing propagation and analysis**
- 05 Timing engine**
- 06 Timing learning**
- 07 Experimental results**
- 08 Future works**

# Background

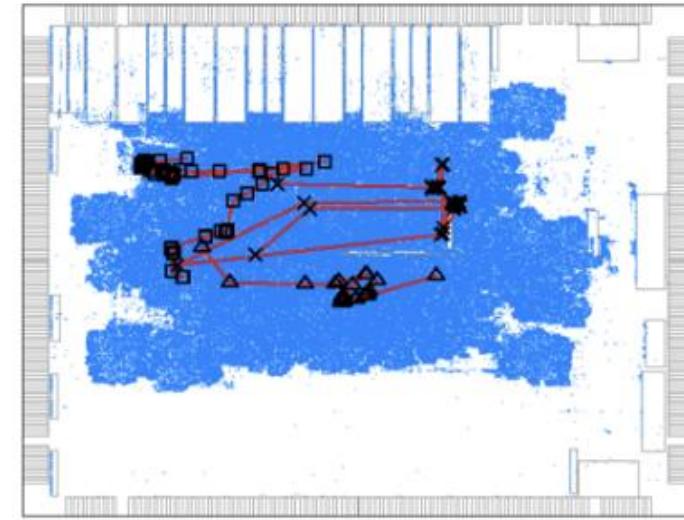
- The critical nature of Static Timing Analysis
  - Determines whether a chip design meets the performance requirements and timing constraints
  - Evaluating the timing of the design is necessary at every stage of chip design
  - Analysis speed and accuracy



EDA tool chain

# Background

- Existing popular open-source STA tools
  - OpenTimer
    - CPP-Taskflow, Parallelization in incremental timing analysis.
    - Award-winning tools and golden timers in CAD Contests
    - CPU-GPU heterogeneous accelerated
  - OpenSTA
    - Part of OpenRoad project
    - More comprehensive functions
    - Autonomous, No-Human-In-Loop (NHIL) flow, 24 hour turnaround



[1] <https://github.com/OpenTimer/OpenTimer>

[2] <https://github.com/The-OpenROAD-Project/OpenSTA>

# Background

- Features comparison among iSTA, OpenTimer and OpenSTA

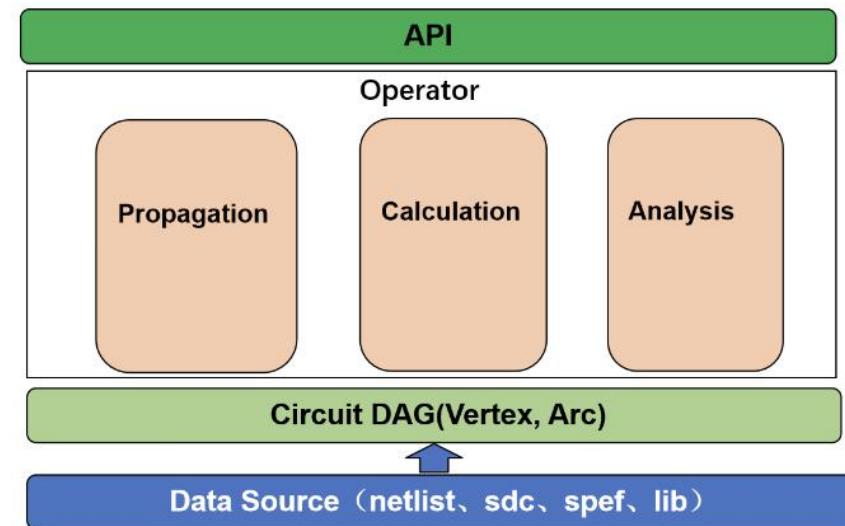
Feature	iSTA	OpenTimer	OpenSTA
Def Input	✓	✗	✓
Setup/Hold Check	✓	✓	✓
NLDM/Elmore	✓	✓	✓
CCS model	✓	✗	✓
High Order Delay Calculation	✓	✗	✓
SDF Annotate	✗	✗	✓
OCV	✓	✓	✓
AOCV	✓	✗	✓
POCV	✗	✗	✓
IRDrop Aware	✗	✗	✗
Hierarchy Netlist Analysis	✗	✗	✗
Crosstalk Analysis	basic	✗	✗
Clock Gate Analysis	✓	✗	✓
Latch Analysis	✗	✗	✓

# Background

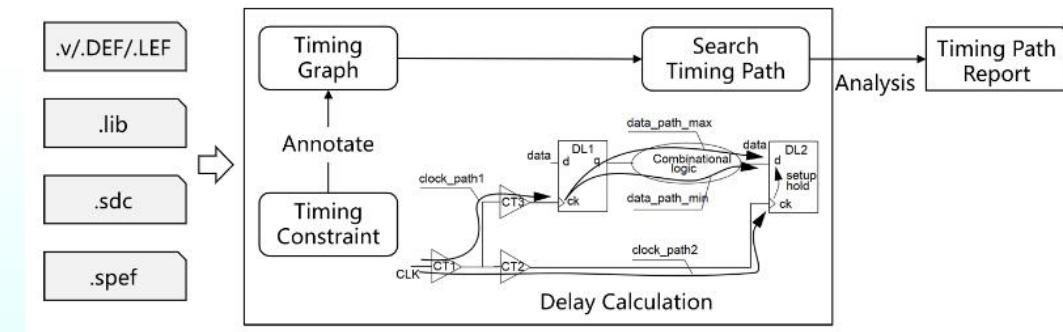
- iSTA concerns
  - Generic timing analysis functions have been implemented.
    - iSTA aims to construct a set of functions and algorithms. Developers or users can customize the implementation of the tool according to their own needs.
  - Separated timing graph data and operation operator
    - Three-tier architecture, with the data layer, operator layer and API layer, the software structure is scalable.
    - Integrated as a timing engine into other tools, provide rich APIs to different stages such as floorplan, placement, CTS, TO, routing, etc.
  - Mixing C++20 and Rust programming paradigms
    - Use Rust to build parsers for basic input files
    - C++20 adopts the latest Range library and other features
  - AI calibrated timing path analysis results
    - Use the AI learning model to replace the interpolation process of cell delay
    - Apply AI model to calibrate the overall path delay

# iSTA structure

- iSTA software architecture
  - Input file parser and database
    - Rust language
  - Graph data
    - Timing graph
  - Operator layer
    - Timing propagation
    - Timing calculation
    - Timing analysis
  - Interface layer
    - Timing engine
    - Python, Tcl, C++



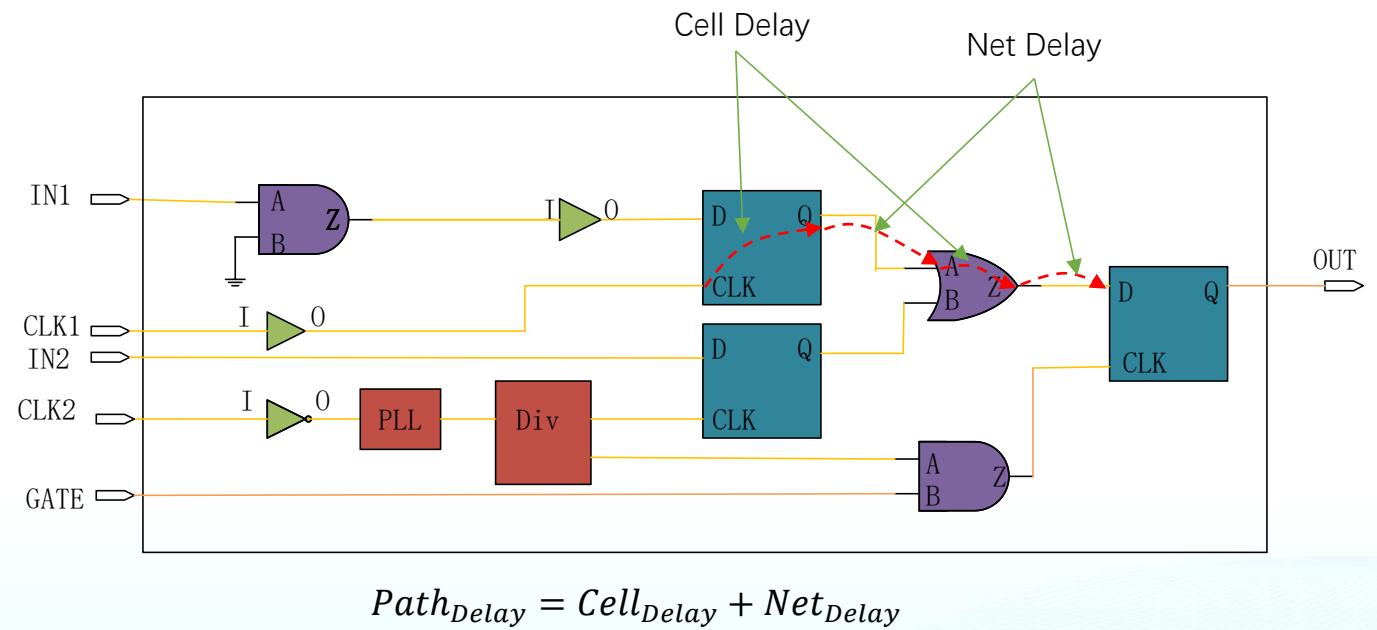
iSTA software structure



STA function

# Timing calculation

- Cell delay
  - NLDM
  - CCS
- Interconnect delay
  - First order moment model
    - Elmore delay metric
  - Second order moment model
    - D2M delay metric
    - ECM delay metric
    - MD2M delay metric
  - High order model
    - Arnoldi delay metric



# Timing calculation

- Cell delay calculation
  - NLDM (Non-Linear Delay Model)
  - Linear interpolation
  - Look up table (LUT)
    - Index1 - input slew  $S_{in}$
    - Index2 – output capacitance  $Cap_{out}$
  - $D_{cell} = D_0 + D_1 * S_{in} + D_2 * Cap_{out}$ 
    - where D0, D1, and D2 are constants

```
pin (OUT) {
    max_transition : 1.0;
    timing() {
        related_pin : "INP";
        timing_sense : negative_unate;
        rise_transition(delay_template_3x3) {
            index_1 ("0.1, 0.3, 0.7"); /* Input transition */
            index_2 ("0.16, 0.35, 1.43"); /* Output capacitance */
            values ( /* 0.16      0.35      1.43 */ \
                     /* 0.1 */ "0.0417, 0.1337, 0.4680", \
                     /* 0.3 */ "0.0718, 0.1827, 0.5676", \
                     /* 0.7 */ "0.1034, 0.2173, 0.6452");
        }
        fall_transition(delay_template_3x3) {
            index_1 ("0.1, 0.3, 0.7"); /* Input transition */
            index_2 ("0.16, 0.35, 1.43"); /* Output capacitance */
            values ( /* 0.16      0.35      1.43 */ \
                     /* 0.1 */ "0.0817, 0.1937, 0.7280", \
                     /* 0.3 */ "0.1018, 0.2327, 0.7676", \
                     /* 0.7 */ "0.1334, 0.2973, 0.8452");
        }
    }
}
```

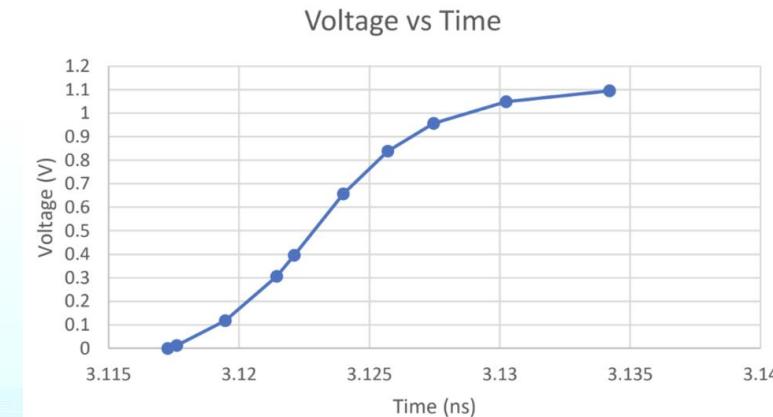
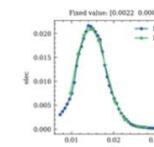
Look up table (LUT)

# Timing calculation

- Cell delay calculation
  - CCS (Composite Current Source)
  - Current source
  - Linear interpolation
  - Look up table (LUT)
    - Index1 – input slew  $S_{in}$
    - Index2 – output capacitance  $Cap_{out}$
    - Index3 – time values

$$i(t) = C(t) \cdot dV(t)/d(t)$$

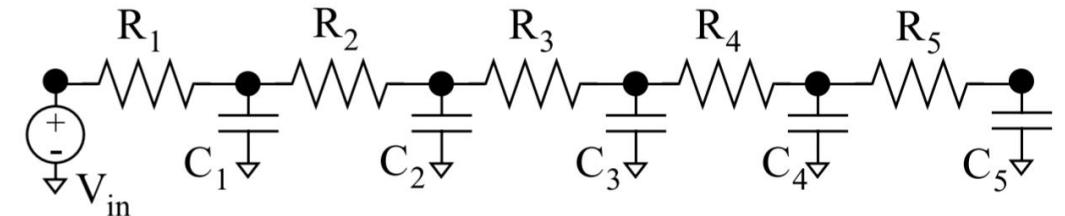
```
pin (OUT) {
    ...
    timing () {
        related_pin : "IN"-;
        ...
        output_current_fall () {
            vector ("LOOKUP_TABLE_1x1x5") {
                reference_time : 5.06; /* Time of input crossing
                threshold */
                index_1("0.040"); /* Input transition */
                index_2("0.900"); /* Output capacitance */
                index_3("5.079e+00, 5.093e+00, 5.152e+00,
                5.170e+00, 5.352e+00");/* Time values */
                /* Output charging current: */
                values("-5.784e-02, -5.980e-02, -5.417e-02,
                -4.257e-02, -2.184e-03");
            }
            ...
        }
        ...
    }
    ...
}
```



# Net delay calculation

- Elmore delay metric
  - Update downstream capacitances of each node
  - Update delay from root to each node
- Characteristics
  - Simple closed form expression

$$\text{delay}_{\text{root} \rightarrow \text{node } j} = \sum_{\substack{\text{nodes } i \\ \text{from} \\ \text{root to node } j}} R_i (\sum \text{downstream caps})$$



A simple RC tree network

- Fast computation speed
- First moment of the impulse response
- Errors are pronounced for near-end nodes
  - Resistance shielding is less dominant for far-end nodes
  - Upper bound on delay

[5] Elmore W C. The transient response of damped linear networks with particular regard to wideband amplifiers[J]. Journal of applied physics, 1948, 19(1): 55-63.

[6] Alpert C J, Devgan A, Kashyap C. A two moment RC delay metric for performance optimization[C]//Proceedings of the 2000 international symposium on Physical design. 2000: 69-74.

# Net delay calculation

- Delay with two moments (D2M)

- Step 1: calculate Elmore delay

$$ED_i = \sum_{k=1}^N R_{ki} C_k$$

- Step 2: Second moment of the impulse response,  $m_0 = 1$

$$m_j^{(i)} = - \sum_{k=1}^N R_{ki} C_k m_{j-1}^{(k)}$$

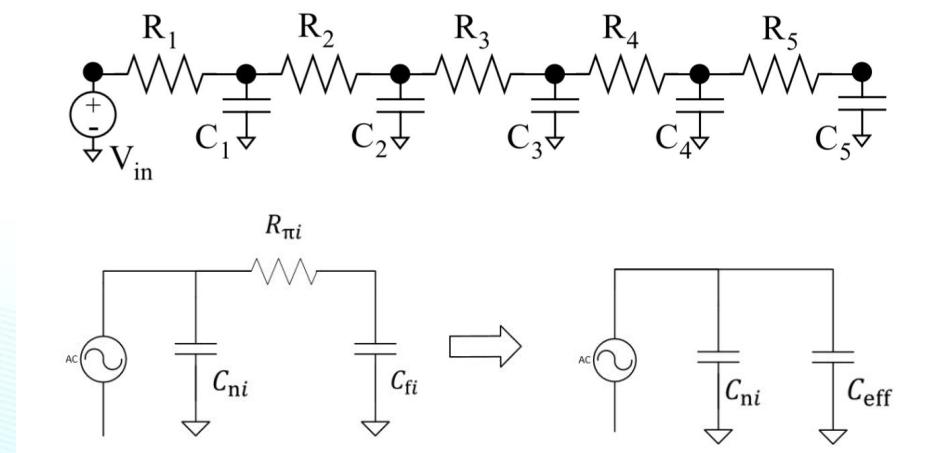
- Step 3:

$$D2M = \frac{m_1^2}{\sqrt{m_2}} \ln 2$$

- As simple and fast as the Elmore delay
  - Significantly more accurate than Elmore
  - Remarkably accurate at the far end of RC lines
  - Ignore resistance shielding

# Net delay calculation

- Effective capacitance metric (ECM)
  - Downstream capacitance is modeled by an effective capacitance instead of the sum of capacitances
  - Consider resistance shielding



Effective capacitance computation based on a reduced model.

Formulated in a Taylor series

Step1: 
$$Y_i(s) = y_{1,i}s + y_{2,i}s^2 + y_{3,i}s^3 + \dots$$

↓

Calculated with the three moments

Step2: 
$$R_{\pi i} = -\frac{y_{3,i}^2}{y_{2,i}^3}, \quad C_{fi} = \frac{y_{2,i}^2}{y_{3,i}}, \quad C_{ni} = y_{1,i} - C_{fi}$$

↓

Effective capacitance

Step3: 
$$C_{eff} = C_{fi} (1 - e^{-ED_i/\tau_i})$$

↓

Effective load capacitance

Step4: 
$$C_{leff} = C_{ni} + C_{eff}$$

[8] Kashyap, C. V, C. J. Alpert, and A. Devgan, "An "effective" capacitance based delay metric for rc interconnect," in IEEE/ACM International Conference on Computer Aided Design. ICCAD-2000. IEEE/ACM Digest of Technical Papers (Cat. No. 00CH37140), pp. 229–234, IEEE, 2000.

# Net delay calculation

- Effective capacitance metric (ECM)
  - Expression

$$D_{ECM} = \sum_{i \in N} (R_i * \sum C_{leff})$$

- Same form as the Elmore formula
- Same complexity as the Elmore delay
- More accurate than Elmore delay
- Do not require the computation of multiple moments

# Net delay calculation

- Modified delay with two moments (MD2M)
  - Expression

$$D_{MD2M} = \frac{m_1'^2}{\sqrt{m_2'}} \ln 2$$

- where  $m'_1$  and  $m'_2$  are the modified first two moments of the impulse response. Utilize the effective capacitance of the downstream capacitances from node i to substitute the sum of downstream capacitances.

- As simple and fast as the D2M delay metric
- Significantly more accurate than D2M
- Remarkably accurate at the far end of RC lines
- Consider resistance shielding

- High order moment delay metric

Modeled as RLC/RC circuit, KVL/KCL

$$\begin{cases} G_x x(t) + C_x \dot{x}(t) = B_x u(t) \\ y(t) = L_x^T x(t) \end{cases}$$



Laplace transform

$$\begin{cases} (G_x + sC_x)x(s) = B_x u(s) \\ y(s) = L_x^T x(s) \end{cases}$$



Transfer function

$$H_f(s) = \frac{y(s)}{u(s)} = L_x^T (G_x + sC_x)^{-1} B_x$$



$$\begin{aligned} A_x &= G_x^{-1} C_x, & R_x &= G_x^{-1} B_x \\ H_f(s) &= L_x^T (I + sA_x)^{-1} R_x \end{aligned}$$

Original circuit system

**Algorithm 1** Arnoldi Algorithm

**Input:**  $A, b$

**Output:**  $V, H$

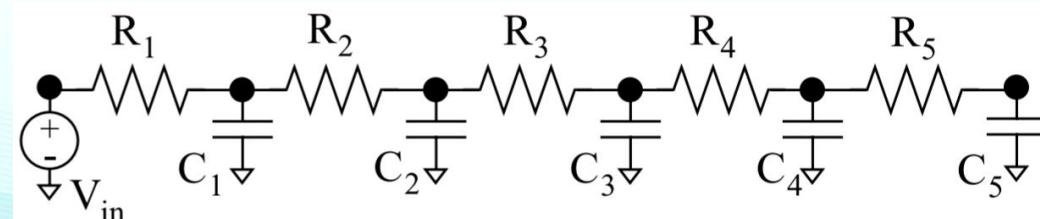
```

1: Compute  $v_1 = \frac{b}{\|b\|_2}$ 
2: for  $j = 1$  to  $q$  do
3:   Compute  $w_j = A \cdot v_j$ 
4:   for  $i = 1$  to  $j$  do
5:      $h_{ij} = v_i^T \cdot w_j$ 
6:      $w_j = w_j - h_{ij} \cdot v_i$ 
7:   end for
8:   if  $w_j = 0$  then
9:     Break and return
10:  else
11:     $h_{j+1,j} = \|w_j\|_2$ 
12:     $v_{j+1} = \frac{w_j}{h_{j+1,j}}$ 
13:  end if
14: end for

```

$$\tilde{G}_x = W^T G_x V, \quad \tilde{C}_x = W^T C_x V, \quad \tilde{B}_x = W^T B_x, \quad \tilde{L}_x = V^T L_x$$

Moment:  $\bar{M}_k = \bar{L}^T \bar{A}^k \bar{R}$



$$\begin{cases} (\tilde{G}_x + s\tilde{C}_x)\tilde{x}(s) = \tilde{B}_x u(s) \\ \tilde{y}(s) = \tilde{L}_x^T \tilde{x}(s) \end{cases}$$

$$\tilde{H}_f(s) = \frac{\tilde{y}(s)}{u(s)} = \tilde{L}_x^T (\tilde{G}_x + s\tilde{C}_x)^{-1} \tilde{B}_x$$

Reduced order system

- Match  $q$  moments for  $q^{\text{th}}$ -order approximation
- Numerical stability
- Passivity preservation

- Elmore metric – First order moment

Transfer function

$$H_f(s) = \frac{y(s)}{u(s)} = L_x^T(G_x + sC_x)^{-1}B_x$$



$$\begin{aligned} A_x &= G_x^{-1}C_x, & R_x &= G_x^{-1}B_x \\ H_f(s) &= L_x^T(I + sA_x)^{-1}R_x \end{aligned}$$



Taylor expansion

$$h(s) = \sum_{k=0}^{\infty} m_k s^k$$

$$m_k = \frac{1}{k!} \times \frac{d^k h(s)}{ds^k} \Big|_{s=0}$$

where the  $k^{\text{th}}$  coefficient of  $h(s)$ ,  $m_k$  is called the  $k^{\text{th}}$  moment.

For the impulse function  $\delta(t)$ , its Laplace transformation is 1. The transfer function is also the impulse response at the port. For impulse function  $h(t)$ , after Laplace transform and Taylor expand

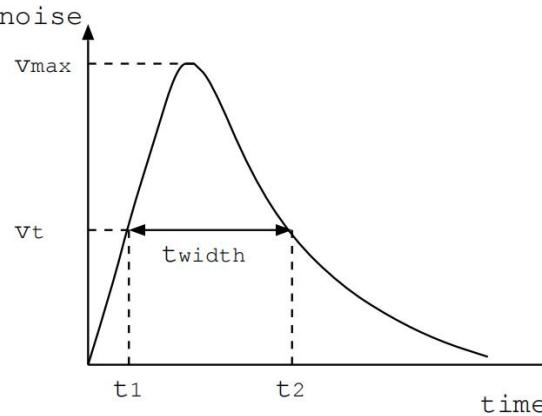
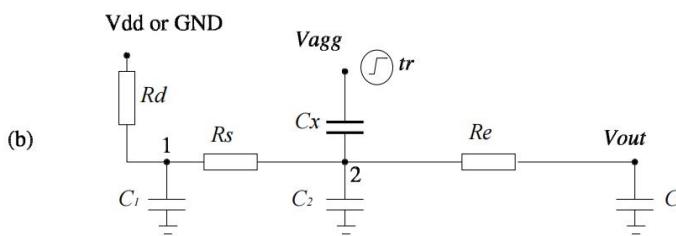
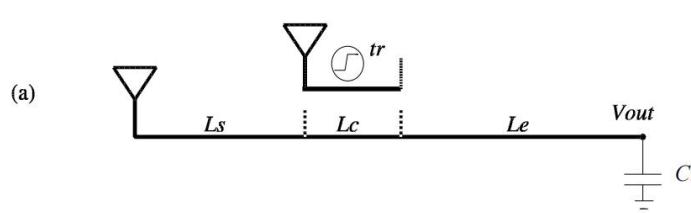
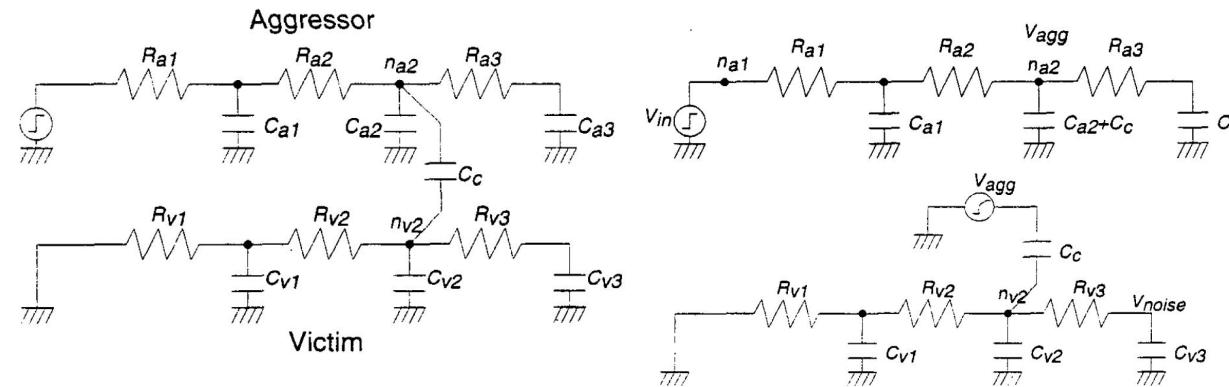
$$h(s) = \int_0^\infty e^{-st} h(t) dt$$

For ideal delay element's transformer function is

$$h_d(s) = e^{-sT}$$

$$T_d = -\frac{d(h_d(s))}{ds} \Big|_{s=0} = m_1$$

# Crosstalk noise calculation



- The impedance at node 1, satisfying the following

$$\frac{1}{Z_1} = \frac{1}{R_d} + sC_1$$

- Then at node 2, we have

$$\frac{1}{Z_2} = \frac{1}{(Z_1 + R_s)} + sC_2 + \frac{1}{R_e + \frac{1}{sC_L}}$$

- Denote the s-domain voltage at node 2 by  $V_2(S)$ , then

$$V_2(s) = \frac{Z_2}{Z_2 + \frac{1}{sC_x}} \cdot V_{agg}(s).$$

- The output voltage  $V_{out}(S)$  in the s-domain is

$$V_{out}(s) = V_2(s) \cdot \frac{\frac{1}{sC_L}}{R_e + \frac{1}{sC_L}}.$$

- Substituting  $Z_1$ ,  $Z_2$  and  $V_2$  into  $V_{out}(S)$ , we have

$$\begin{aligned} V_{out}(s) &= \frac{Z_2}{Z_2 + (\frac{1}{sC_x})} \cdot \frac{1/sC_L}{R_e + 1/sC_L} \cdot V_{agg}(s) \\ &= \frac{a_2 s^2 + a_1 s}{s^3 + b_2 s^2 + b_1 s + b_0} \cdot V_{agg}(s) \end{aligned}$$

# Crosstalk noise calculation

- For the aggressor with saturated ramp input with normalized  $V_{dd}=1$  and transition time  $t_r$

$$v_{agg}(t) = \begin{cases} t/t_r & 0 \leq t \leq t_r \\ 1 & t > t_r, \end{cases}$$

- Its Laplace transformation is

$$V_{agg}(s) = \frac{1 - e^{-st_r}}{s^2 t_r}$$

- Using dominant-pole approximation method

$$V_{out}(s) \approx \frac{a_1 s}{b_1 s + b_0} \cdot V_{agg}(s) = \frac{t_x(1 - e^{-st_r})}{s t_r (s t_v + 1)}$$

- Computing the inverse Laplace transform, time domain waveform

$$v_{out}(t) = \begin{cases} \frac{t_x}{t_r} (1 - e^{-t/t_v}) & 0 \leq t \leq t_r \\ \frac{t_x}{t_r} (e^{-(t-t_r)/t_v} - e^{-t/t_v}) & t > t_r \end{cases}$$

$$v_{max} = \frac{t_x}{t_r} (1 - e^{-t_r/t_v})$$

- $v'_{max}$  is indeed a first-order approximation of  $v_{max}$

$$\begin{aligned} \frac{t_x}{t_r} (1 - e^{-t_r/t_v}) &= \frac{t_x}{t_v} [1 - \frac{1}{2} \frac{t_r}{t_v} + \dots] \\ &\approx \frac{t_x}{t_v} \frac{1}{1 + \frac{1}{2} \frac{t_r}{t_v}} = \frac{t_x}{t_v + t_r/2} \end{aligned}$$

- Noise Width: Given certain threshold voltage level  $v_t$ , the noise width for a noise pulse is defined to be the length of time interval that noise spike voltage  $v$  is larger or equal to  $v_t$ .

$$t_2 - t_1 = t_v \ln \left[ \frac{(t_x - t_r v_t)(e^{t_r/t_v} - 1)}{t_r v_t} \right]$$

$$v_t = v_{max}/2$$

$$t_{width} = t_2 - t_1 = t_r + t_v \ln \left[ \frac{1 - e^{-2t_r/t_v}}{1 - e^{-t_r/t_v}} \right]$$

# Timing propagaton

- The calculation and propagation of slew/delay along the timing path
  - Arrival time
  - Required arrival time
  - Forward propagation
  - Backward propagation
  - Timing path
    - Clock path
    - Data path
  - DFS

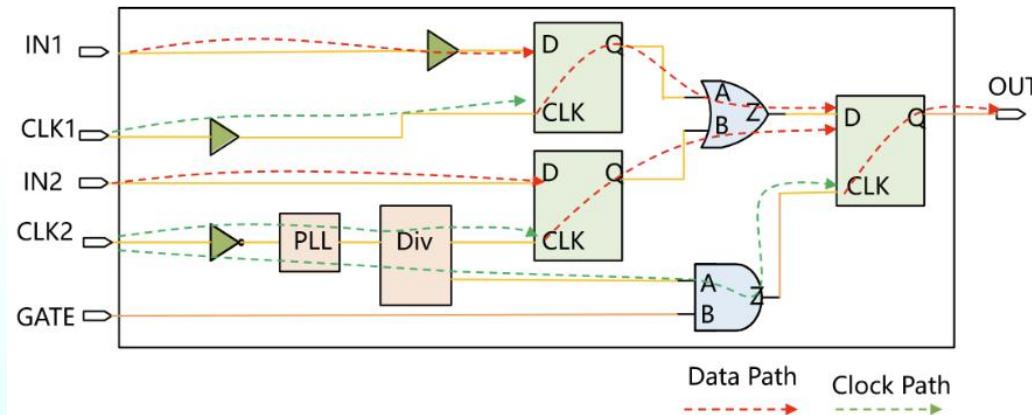
---

**Algorithm 1** Timing Propagation (DFS) Algorithm

---

```
1: Input:  $G$  Timing Graph
2: Input:  $e$  Timing Path end vertex
3: Output:  $V$  visited nodes set
4: function DFS( $G, v, V$ )
5:   Add  $v$  to  $V$ :  $V \leftarrow V \cup \{v\}$ 
6:   for all unvisited neighbor  $u$  of  $v$  do
7:     if  $u$  is start vertex then
8:       init slew or delay or AT data
9:     else
10:      if  $u$  is not yet visited then
11:        Recursively call DFS to visit  $u$ : DFS( $G, u, V$ )
12:        calc slew or delay or AT data use  $v$ 
13:      else
14:        return true
15:      end if
16:    end if
17:   end for
18: end function
```

---



Timing Propagation

# Timing analysis

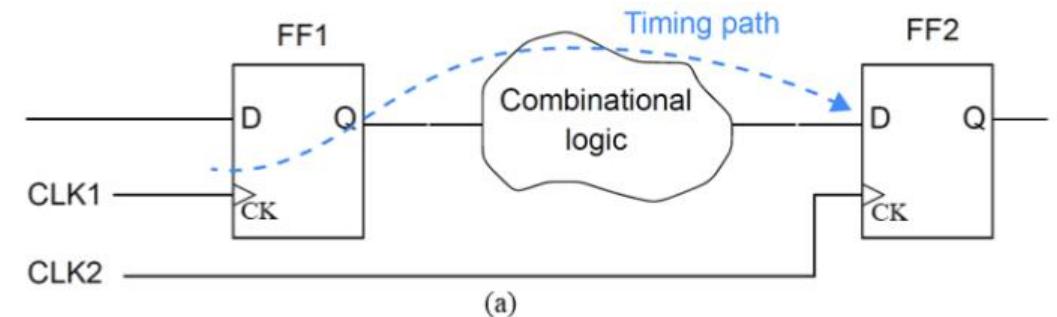
- Setup/hold timing check

- Setup check

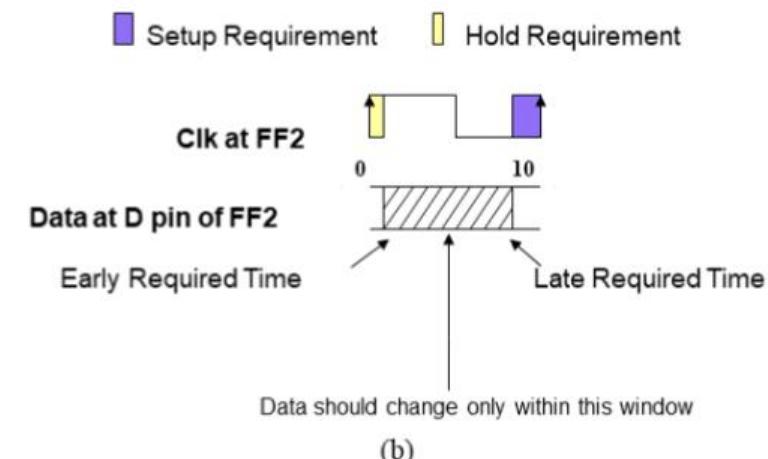
$$T_{\text{launch}} + T_{\text{ck2q}} + T_{dp} < T_{\text{capture}} + T_{\text{cycle}} - T_{\text{setup}}$$

- Hold check

$$T_{\text{launch}} + T_{\text{ck2q}} + T_{dp} > T_{\text{capture}} + T_{\text{hold}}$$



(a)



(b)

Setup/Hold timing check of a flip-flop

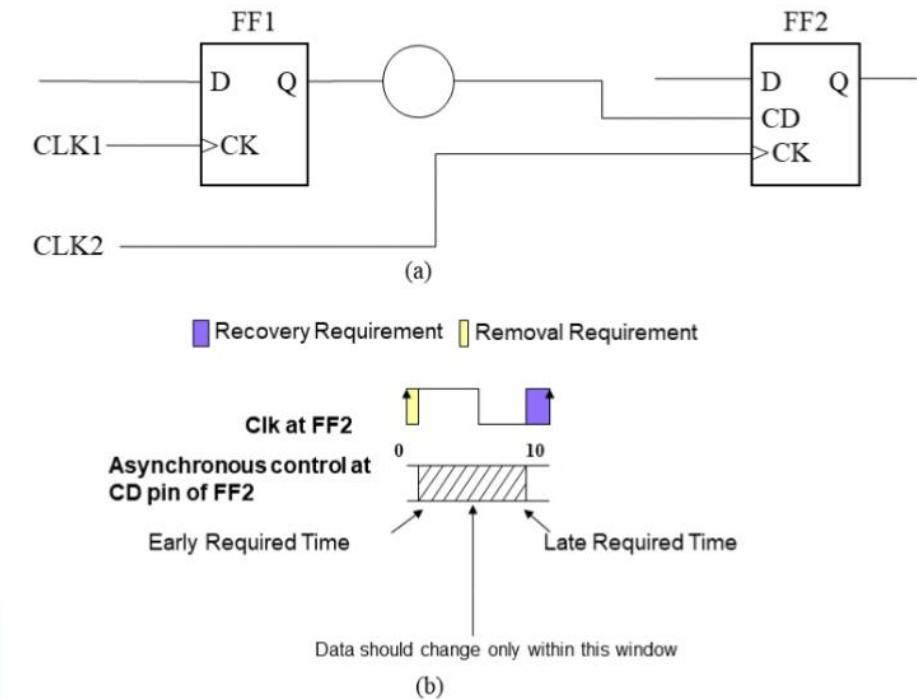
# Timing analysis

- Recovery check
  - Ensures that there is a minimum amount of time between the asynchronous signal(the asynchronous control pin("CD") pin of FF2) becoming inactive and the next active clock edge(the clock pin("CK") of FF2).

$$T_{\text{launch}} + T_{\text{ck2q}} + T_{q2cd} < T_{\text{capture}} + T_{\text{cycle}} - T_{\text{recovery}}$$

- Removal check
  - Ensures that there is adequate time between an active clock edge(the clock pin("CK") of FF2) and the release of an asynchronous control signal(the asynchronous control pin("CD") pin of FF2)

$$T_{\text{launch}} + T_{\text{ck2q}} + T_{q2cd} > T_{\text{capture}} + T_{\text{removal}}$$



Recovery/Removal timing check of a flip-flop

# Timing analysis

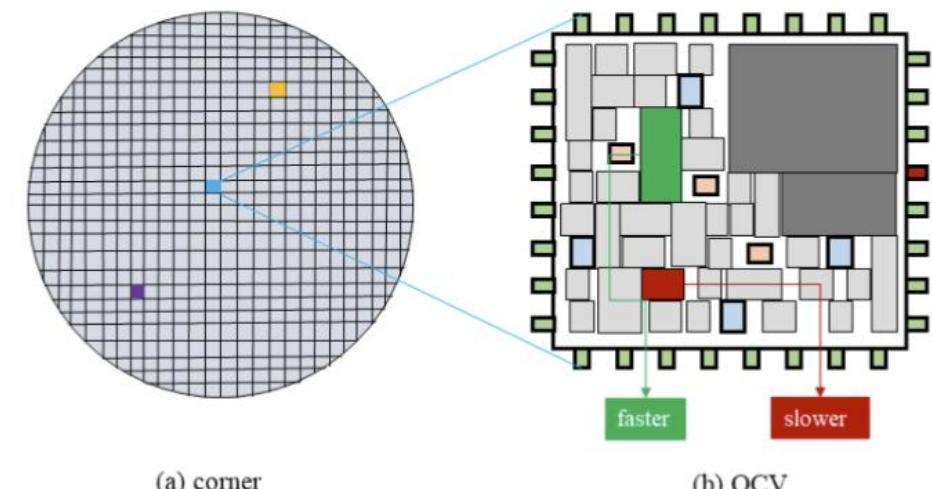
- Multicycle paths
  - The combinational data path between two flip-flops can take more than one clock period to propagate through the logic.
  - The combinational path is declared as a multicycle path by user using the sdc command set multicycle path
  - Sdc command
    - Set\_multicycle\_path 3 -setup -from [get\_pins FF1/Q] -to [get\_pins\_FF2/D]

$$T_{\text{launch}} + T_{\text{ck2q}} + T_{dp} < T_{\text{capture}} + 3 * T_{\text{cycle}} - T_{\text{setup}}$$

# Timing analysis

- On chip variations

- The process(P) and environmental parameters may not be uniform across different portions of the die
- Different portions of the design may also see different power supply voltage(V) and temperature(T)
- Two regions of the same chip are not at identical PVT conditions, the variations in PVT conditions can affect the wire delays and cell delays in different portions.
- Process variations
- Environmental variations
- Aging effects



On-Chip Variations

# Timing analysis

- On chip variations

- OCV

- It assumes that the timing delays of all cells in the circuit are affected by a uniform derating factor, which represents the maximum amount by which the delays can deviate from their nominal values.

- AOCV

- Takes into account the spatial distribution of process variations. It assumes that delays are more affected near the edges of the chip and in hotter regions due to increased sensitivity to process variations.

- POCV

- It considers the statistical distribution of delay variations for each cell based on its specific parameters, such as transistor sizes and doping levels.

Feature	OCV	AOCV	POCV
Modeling complexity	Simple	Moderate	Complex
Accuracy	Less accurate	More accurate	Most accurate
Computational cost	Low	Moderate	High

# Timing engine

- Top-level interface of iSTA tool
  - Powerful C++ APIs
    - Read files
    - Build\_RC tree
    - Operate\_staGraph
    - Modify\_design
    - Update\_timing
    - Report methods
    - Checker methods

TABLE 6: The reader methods.

Method	Description
readLiberty	read the liberty files.
readDesign	read the design verilog file.
readSpef	read the spef file.
readSdc	read the sdc file.
readAocv	read the aocv files.

TABLE 7: The build\_RCtree methods.

Method	Description
makeOrFindRCTreeNode	make the RCtree's node.
incrCap	set the node's cap.
makeResistor	make resistor edge of the RCtree.
buildRCTree	build the RCtree according to the spef file.
updateRCTreeInfo	update the RC info after make the RCtree.
initRcTree	init one RCtree.
resetRcTree	reset the RCtree to nullptr.

TABLE 8: The operate\_staGraph methods.

Method	Description
buildGraph	build the graph data.
isBuildGraph	judge whether it has build graph.
resetGraph	reset graph.
resetGraphData	reset graph data.

# Timing engine

- Top-level interface of iSTA tool

TABLE 9: The modify\_design methods.

<b>Method</b>	<b>Description</b>
insertBuffer	insert buffer need to change the netlist.
removeBuffer	remove buffer need to change the netlist.
repowerInstance	change the size or level of an existing instance.
moveInstance	move the instance to a new location.
writeVerilog	write verilog file according to the netlist.

TABLE 10: The update\_timing methods.

<b>Method</b>	<b>Description</b>
incrUpdateTiming	incremental propagation to update timing data.
updateTiming	update timing data.

TABLE 11: The reporter methods.

<b>Method</b>	<b>Description</b>
reportTiming	generate the timing report.
setSignificantDigits	set the significant digits of timing report.
reportSlew	report the slew of the pin.
reportAT	report the arrival time at a pin.
reportRT	report the required arrival time at a pin.
reportSlack	report the slack at a pin.
reportWNS	report the worst negative slack of the clock group path.
reportTNS	report the total negative slack of the clock group path.
reportClockSkew	report the skew between two clocks.
reportInstDelay	report instance delay.
reportInstWorstArcDelay	report the worst arc delay for the specified instance.
reportNetDelay	report net delay.
...	...

# Timing engine

- Perform iSTA in python iterative mode and TCL iterative mode

TABLE 14: The example of using the iSTA tool in Python interactive mode.

---

**The example of using the iSTA tool in Python interactive mode.**

---

```
ista_cpp.set_design_workspace(work dir +"/rpt")
ista_cpp.read_netlist(work dir +"/example1.v")
ista_cpp.read_liberty([work dir +"/example1_slow.lib"])
ista_cpp.link_design("top")
ista_cpp.read_sdc(work dir +"/example1.sdc")
ista_cpp.read_spf(work dir +"/example1.spf")
ista_cpp.read_lef_def(["example1.tlef","example1.lef"],"example1.def")
ista_cpp.report_timing()
```

---

TABLE 13: The example of using the iSTA tool in TCL interactive mode.

---

**The example of using the iSTA tool in TCL interactive mode.**

---

*First, write the run\_ista.tcl;*

---

```
set work_dir "../src/operation/iSTA/source/data/example1"
set_design_workspace $work_dir/rpt
read_netlist $work_dir/example1.v
set LIB_FILES $work_dir/example1_slow.lib
read_liberty $LIB_FILES
link_design top
read_sdc $work_dir/example1.sdc
read_spf $work_dir/example1.spf
report_timing
```

---

*Second, source the run\_ista.tcl.*

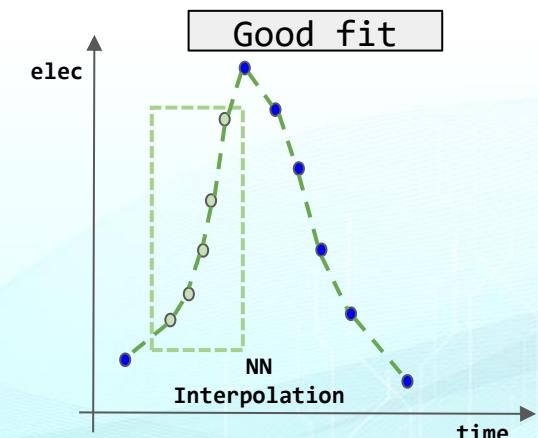
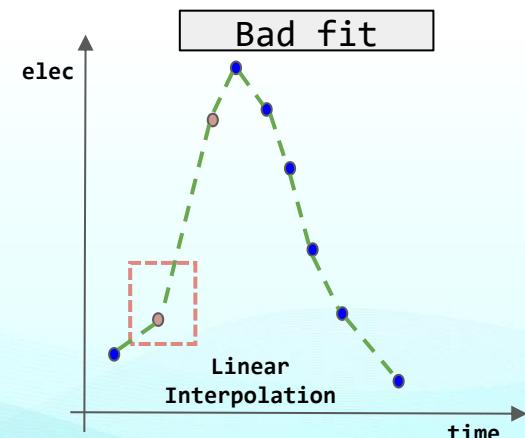
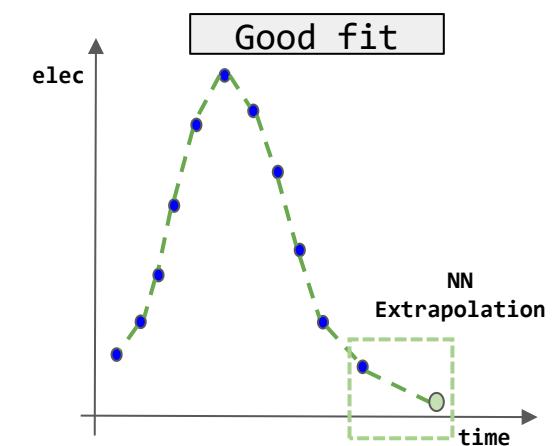
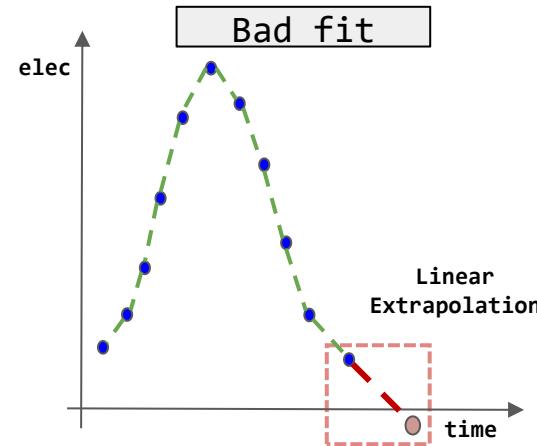
---

```
cd bin/
./iSTA run_ista.tcl
```

---

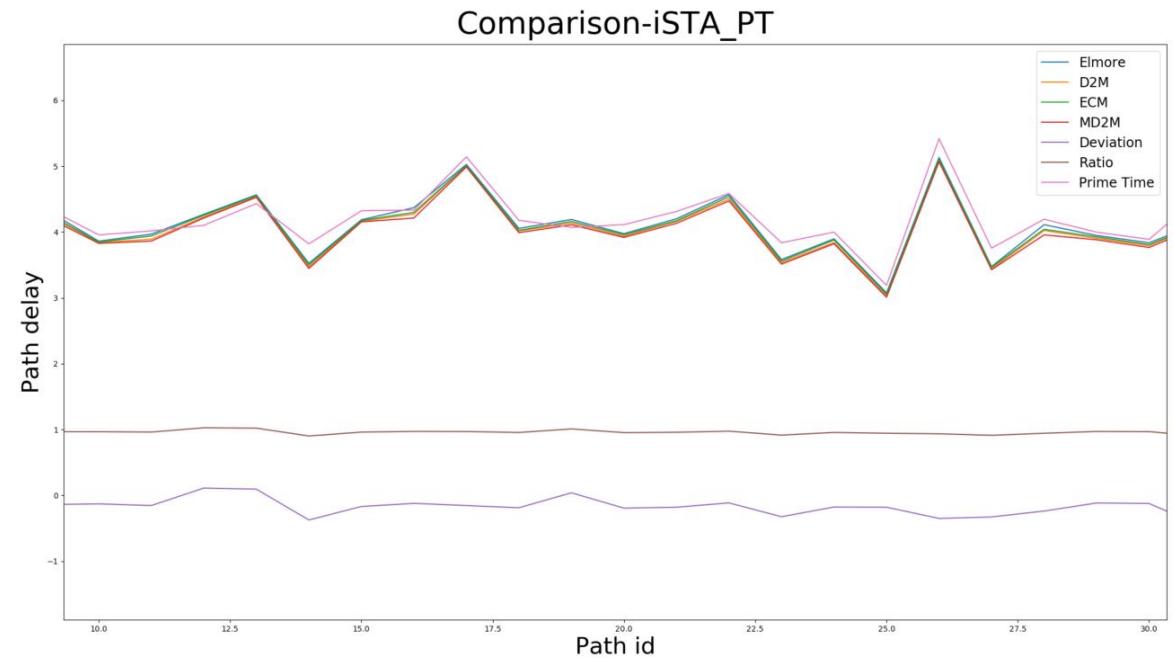
# Timing learning

- Cell delay interpolation learning
  - Instability of extrapolation
  - Uneven distribution of difference points



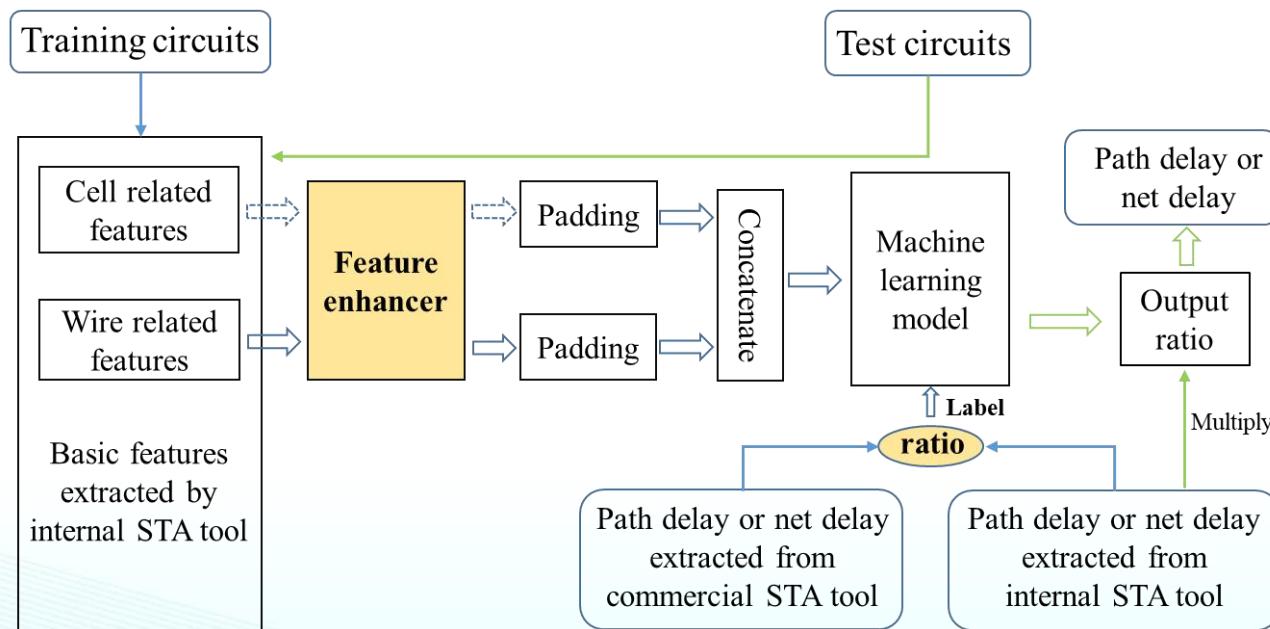
# Timing learning

- Timing path delay learning
  - Lower-order moment delay metrics
    - Efficient and easily computable
    - Less accurate
  - Spice simulation
    - Time-consuming
  - Machine learning model



# Timing learning

- Overflow

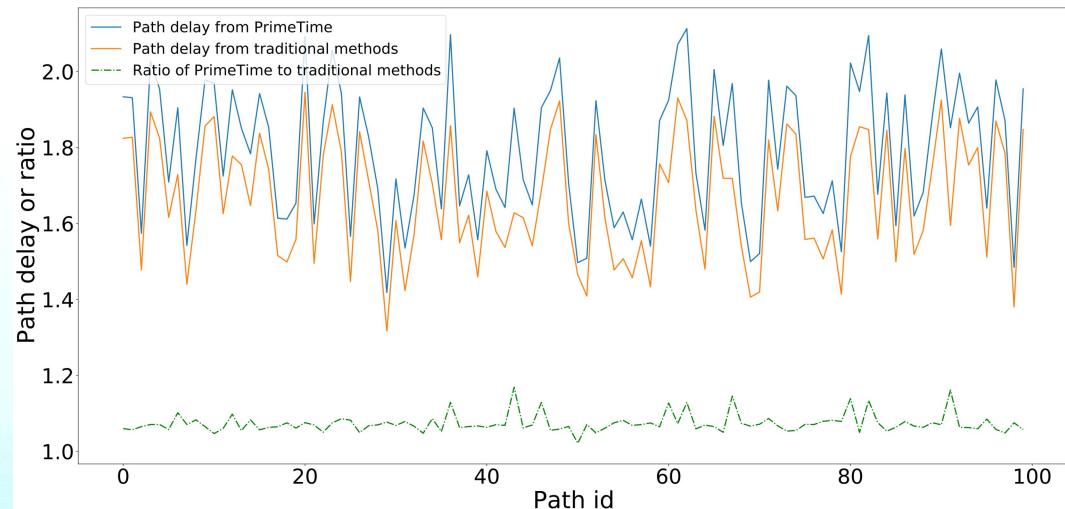


- Features extraction
  - iEDA-iSTA internal tool
  - Label extraction
    - Prime Time
  - Model selection
    - Transformer+MLP
  - Feature enhancer
    - Resistance shielding
    - Effective capacitance
    - Ecm, MD2M
  - Trick
    - Ratio of ground truth to traditional methods

# Timing learning

- Timing path delay learning
  - Feature selection
    - Input slew
    - Load capacitance
    - Output slew
    - Signal polarity
    - Cell type
    - Cell port name
    - Arrive time
      - Cell delay
      - Wire delay (Elmore)
  - Label selection
    - $PD_{PT}$
    - $\frac{PD_{PT}}{PD_{IE}}$
- Experimental results
  - For open-source designs under skywater130 nm process node, and as for the slow corner, the performance of our model with feature enhancer in terms of average rRMSE values are 1.1% and 2.5% for trained designs and unseen designs.

$$PD_{\text{pred}} = f(S_i, SP_{r/s}, CT, CP, S_o, C_l, PD_{FE}) \quad (9)$$



# Timing learning

- Wire delay learning
  - Feature selection

Category	Features
Basic features	Port name
	Resistance
	Capacitance
Feature enhancer_1	Elmore delay
	D2M delay
Feature enhancer_2	Elmore delay
	D2M delay
	ECM delay
	MD2M delay

- Label selection

- $D_{Golden}$
- $\frac{D_{Golden}}{D_{Avg}}$
- Choose the ratio  $D_{Golden}/D_{Avg}$  of golden wire delay  $D_{Golden}$  to average wire delay  $D_{Avg}$  of the four methods above as the label.

$$D_{pred} = f(D_{Elmore}, D_{D2M}, D_{ECM}, D_{MD2M}, D_{Avg})$$

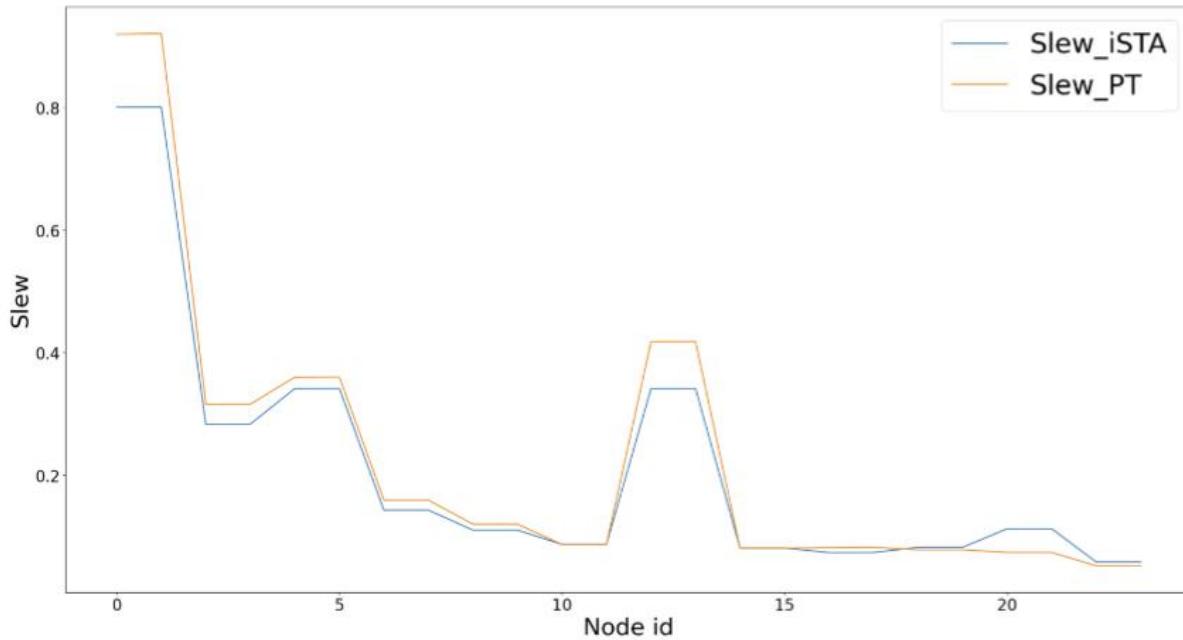
Where  $D_{pred}$  is the predicted wire delay.

- Experimental results

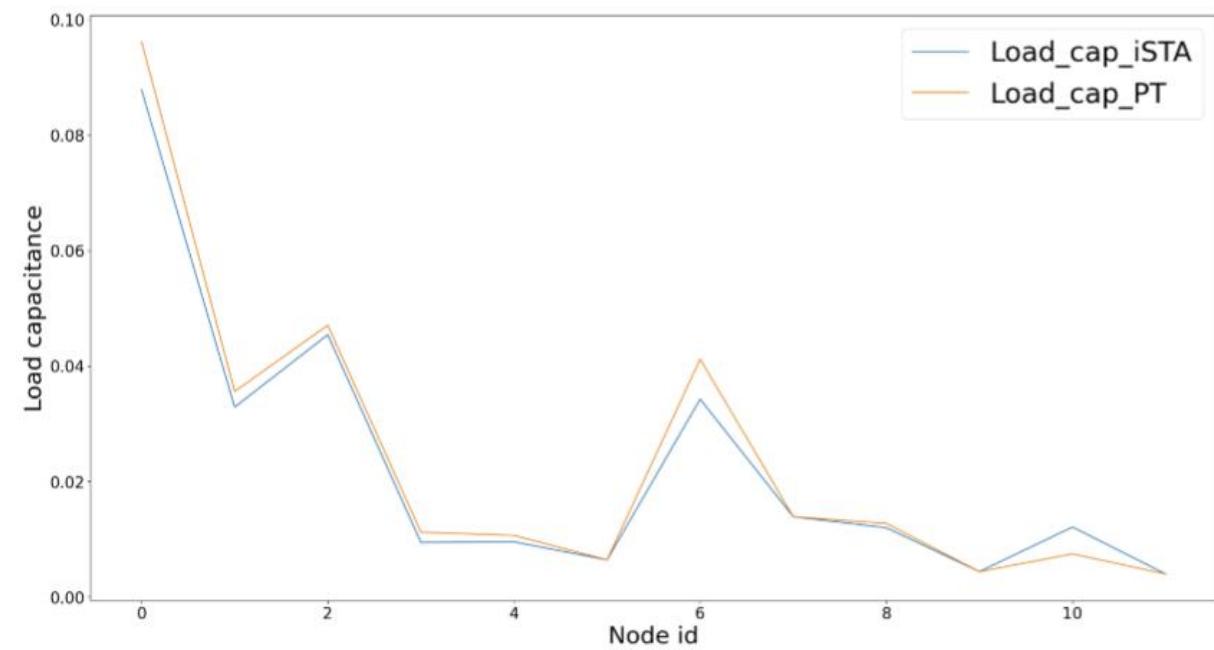
- For wire delay learning of multi-corners with 28nm technology, the model achieves an average rRMSE value of 0.4% for trained designs and 3.3% for unseen designs.

# Results

- Comparision between iSTA and PrimeTime



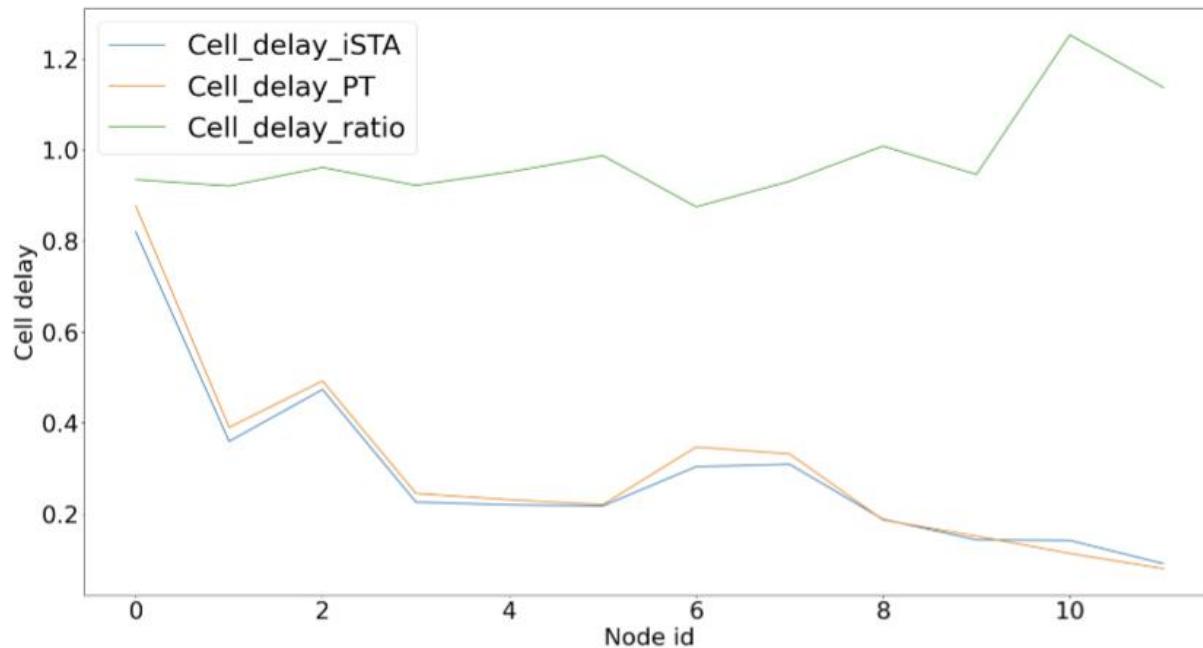
The comparison on the **slew** of timing path between iSTA tool and PrimeTime



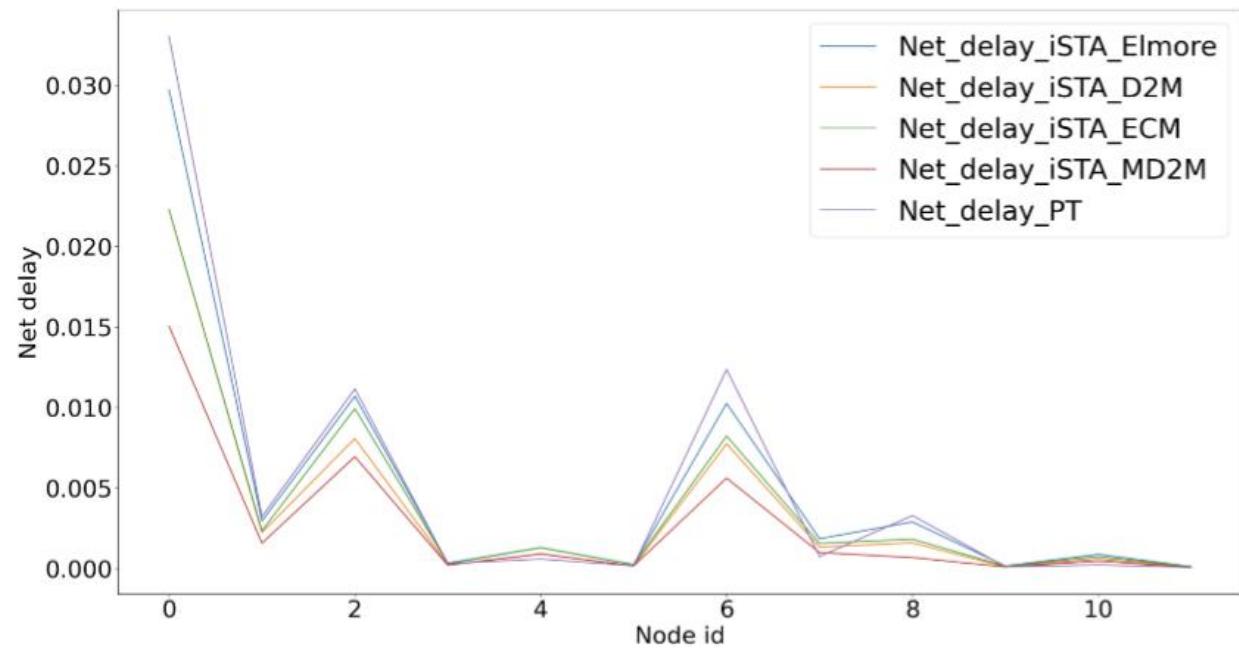
The comparison on the **load capacitance** of timing path between iSTA tool and PrimeTime

# Results

- Comparision between iSTA and PrimeTime



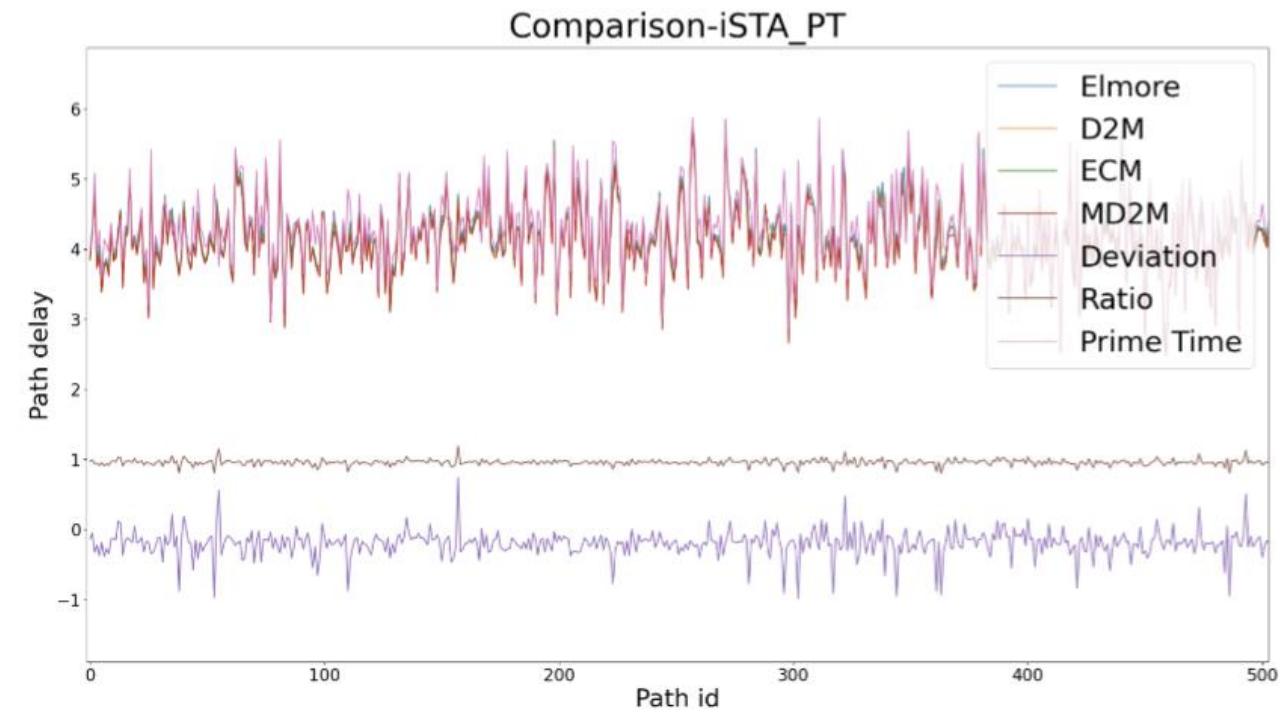
The comparison on the **cell delay** of timing path between iSTA tool and PrimeTime



The comparison on the **net delay** of timing path between iSTA tool and PrimeTime

# Results

- Comparision between iSTA and PrimeTime



The comparison on the **path delay** of timing path between iSTA tool and PrimeTime

# Future works

- Software
  - Architecture more modular, pluggable, and scalable
  - Use Rust programming language
    - More flexible
    - Easy to extend
- Flow
  - Adapt to the development of AI
  - Integrate AI models at various point during the process and store data in checkpoints

# Future works

- Flow
  - AI for STA
    - AI for timing graph reduce
      - AI can determine the impact level of graph nodes based on certain features, thus deciding whether reduction is possible
    - AI for calculation method decision
      - AI can help decide which method to use based on the criticality of the path, reducing the need for manual adjustments after adopting a single strategy, and speeding up the turnaround time.
    - AI for path delay calibration
      - Using AI methods can further calibrate based on mathematical theory calculations
  - STA for AI
    - STA big database for AI
    - A large amount of computing data can be generated in the STA calculation process
    - For the same technology node, many path characteristics may be homogeneous, can be used to predict timing situation, providing an analysis engine for physical design optimization.
    - Provide a large amount of data for large circuit models (LCM)

# Future works

- Technologies
  - AI/ML
    - The idea of AI-native LCMs opens up exciting possibilities in EDA
    - Rust for high performance
      - Rust reduces the risk of data race in multithreading by checking the dependencies between data statically
      - Rust's functional programming approach, combined with crate composition, can reduce the coupling between modules, enhance program scalability, and further improve program parallelism in multi-core environments.

# Conclusions

- Summaries about iSTA 1.0
  - Implements commonly used timing analysis functions
  - The interface supports TCL, python and C++ interfaces
  - Use mixed programming of Rust and C++
  - The design goals are to achieve decoupling, modularization, and easy to use
  - In the future, deeply integrate AI/ML technology and Rust/C++ parallel programming to improve accuracy and running speed of timing analysis.

The screenshot shows the GitHub repository page for the OSCC-Project / iEDA project. The repository has 2,107 commits and 231 stars. The 'About' section notes 'No description, website, or topics provided.' The 'Releases' section indicates 'No releases published' and 'Create a new release'. The 'Contributors' section lists 25 contributors, with a link to '+ 11 contributors'. The 'Languages' section is not visible in the screenshot.



# Thanks

He Liu  
liuh@stu.pku.edu.cn