Department of Electrical and Computer Engineering
University of Thessaly

ECE431 - CAD Algorithms

Fall Semester — Academic Year 2025-2026

Assignment 3rd

Delay Annotation using LUTs

19/11/2025 έως 07/12/2025

C. Sotiriou

Contents

1 Assignment Description

In this assignment, you will extend your previous topological traversal and unit-delay annotation framework by performing accurate gate-level delay annotation using timing data extracted from Liberty standard format library (.lib) file. Gates will be annotated with delays interpolated and extrapolated from the cell lookup tables (LUTs) specific to each cell type and load/slew conditions. As part of this assignment you will identify and report the critical path of the design, in terms of the maximum arrival time (worst-case analysis). The process will follow a graph-based approach in topological order, without considering interconnect delays.

Through this assignment, students gain direct experience in leveraging industry-standard Liberty timing libraries, extracting and utilising Non-Linear Delay Model (NLDM) data, and implementing robust graph-based timing propagation algorithms essential to modern VLSI design flows.

2 Background

In digital circuit design and analysis, accurate timing annotation is essential for performance optimisation and correct functionality. Standard-cell libraries, distributed in Liberty (.lib) format, play a crucial role by providing detailed timing, power, and functional information for each logic gate, flip-flop, or memory element in the design.

Instead of assuming fixed delays for each cell (approach adopted for the 2nd assignment) modern Static Timing Analysis (STA) relies on delay models like the Non-Linear Delay Model (NLDM), where delay values are specified in multi-dimensional lookup tables (LUTs) within Liberty files. For every timing arc – capturing the signal path from a particular input to an output pin – the Liberty format tabulates delays as functions of input transition time (slew) and output capacitance (load). These two-dimensional tables offer a realistic way to model the impact of electrical characteristics on timing, accommodating different operating conditions and cell behaviours. More information regarding the NLDM liberty format and LUTs can be found in Appendix A.

During delay annotation, the algorithm traverses the digital circuit, represented as a directed acyclic graph (DAG), in topological order, ensuring the output pin of each gate is processed after its inputs. For each standard cell, the specific input arrival time and output load under examination determine the delay for that instance, which is retrieved from the NLDM LUT. Usually, the related values under examination (input transition and output load) rarely match the discrete index values in the table, thus, **interpolation**, and sometimes **extrapolation**, techniques must be applied to estimate the delay between tabulated indexes. An example of linear interpolation and extrapolation is provided in Appendix B.

Liberty timing arcs may include "when" conditions, which specify that certain timing characteristics – such as delay and transition time – apply only if particular Boolean expressions on the gate's input pins are satisfied. For instance, the delay from an input to an output might be different if some other input is held high rather than low, reflecting real circuit behaviour under different logical scenarios. When performing timing analysis, the STA engine must evaluate the logic state of the relevant pins for each cell instance as it propagates signal. If the specified "when" condition for a timing arc evaluates to true (matches the current context of the logic simulation or analysis), then the timing data from that arc – including the specific lookup table values for delay and transition – are selected for the calculation. If the condition does not hold, the arc is either ignored or a separate timing entry (possibly the unconditional/default one) is chosen. In summary, "when" conditions dictate both which timing arc is relevant and which LUT from the Liberty file should be consulted for a specific input-to-output timing calculation, allowing for more accurate, scenario-dependent timing annotation in realistic circuit analyses.

In addition to gate delay, interconnect delay, which refers to the propagation time through wires connecting gates, can have a significant impact on the overall timing of digital circuits, especially as designs grow more complex and operate at higher speeds. In advanced, technologies, interconnect delay may approach or even exceed the delays of individual logic gates due to resistance, capacitance, ands coupling effects. Proper modeling of interconnect is essential for predicting critical paths, understanding timing bottlenecks, and ensuring reliable operation of integrated circuits under real-world conditions. However, incorporating interconnect annotation requires additional complexity: accurate parasitic extraction, wire modeling, and consideration of layout-dependent effects. Therefore, for the purposes of this assignment, **the interconnect delay annotation is excluded**. The timing calculations will be based solely on cell-level delay data, assuming ideal interconnects without additional delay contributions from wires.

This delay annotation across gates enables the identification of the circuit's critical path, i.e. the longest timing path from any starting points (primary inputs or sequential elements output pins) to any endpoints (primary outputs or sequential elements input pins) that ultimately determines the minimum achievable clock period for synchronous operation. For the scope of this assignment, **the critical path is characterised by the maximum arrival time observed among all endpoints** after propagating delays through all gates and interconnects (all timing arcs). The critical path can be extracted by retrieving the endpoint with the highest arrival time and then tracing back through the sequence of gates and inputs responsible for this delay. Reporting the critical path offers essential insight into timing bottlenecks and guides design optimisation efforts to meet performance goals.

3 Assignment Guidelines

- **Integration with Topological Traversal**

  Delay annotation must be implemented as part of the topological traversal algorithm developed in previous assignment. This ensures that gates are processed strictly after all their input predecessors are handled, preserving signal dependency ordering. By incorporating delay computations within this traversal, the algorithm efficiently propagates timing information in a single pass without multiple redundant computations.

- **TCL Commands for Initial Conditions**

  Before starting delay annotations, you must implement TCL commands that allow specifying:

  - **Input Transition** (`set_input_transition`) **and Delay** (`set_input_delay`) **for Primary Inputs**:

    Commands to assign the input transition time (input slew) and initial delay values to each primary input pin. These values serve as initial conditions for the delay annotation performed during the topological circuit traversal. Usually, we set these values to 0, but user should be able to specify values for IOs of topmodule.

  - **Output Load** (`set_output_load`) **for Primary Outputs**

    Command to assign output load capacitances to all primary output pins. These loads provide boundary conditions for gates driving the primary outputs and are critical for accurate delay computation during the traversal.

  These commands should support as input either a single port, a list of ports, or the commands **all_inputs_cad1** and **all_outputs_cad1**. The command all_inputs returns a list of all primary inputs of the top module, while all_outputs returns a list of all primary outputs.

  Examples of valid usage:

  - `set_input_delay 0.0 [all_inputs_cad1]`
  - `set_output_load 0.0 [all_outputs_cad1]`
  - `set_input_transition 0.0 {IN1 IN3 IN5}`

- **Delay Computation for Each Gate**

  For every gate visited during the traversal, compute its output delay based on three key inputs:

  - **Input Arrival Times**:

    Obtain the arrival times (delays) from all driving gates connected to the current gate's input pins.

  - **Output Capacitance (Load)**:

    Determine the output load capacitance for the gate by summing the input capacitances of all gates driven by this gate's output pin. An input pin will have rise_capacitance (min, max) and fall_capacitance (min, max).

When we calculate the rise case, we obtain the maximum rise_capacitance value, similarly, for the fall case, we obtain the maximum fall_capacitance value. In most libraries, input pins do not have a capacitance range, so the minimum and maximum values are typically identical.
For primary outputs, use the load specified via the TCL command `set_output_load`.

- **Liberty LUT Delay Values**:
  For each gate, the delay must be obtained from the Liberty LUTs. Each LUT specifies delay values as a function of two electrical parameters:
  * The input transition time (also known as input slew), and
  * The output load capacitance

  If the required input slew or output load for a gate does not exactly match any LUT entry, use linear interpolation between the nearest available values in the table to estimate the delay. If the required value lies outside the provided range in the LUT, perform linear extrapolation using the boundary values.

In static timing analysis (STA), for gates with multiple input-to-output arcs, the worst-case delay is computed by evaluating the delay for each input-to-output arc individually and then selecting the maximum delay among them. This process accounts for the fact that different input signals may arrive at different times and that the propagation delay through the gate can vary depending on which input path is considered.

Specifically:

- For each input pin that has a timing arc to the output pin, compute the delay by summing the arrival time at the input pins and the gate delay associated with the corresponding input-to-output arc. The gate delay is obtain from the Library LUTs via interpolation/extrapolation, as mentioned above, for the given input slew and output load.
- Each input-to-output arc produces one delay value representing the time for a signal arriving on that input to propagate through the gate to the output.
- The gate's output delay is the maximum of all these computed input-to-output arc delays. The max operation ensures that the timing analysis considers the worst-case signal arrival and gate delay combination, which is essential for accurate timing verifications.

This max delay selection effectively models worst-case timing behaviour in gates with multiple inputs, aligning with the fundamental principles of static timing analysis.

More information regarding the format of the NLDM Liberty Format, LUTs and how to access them can be found in Appendix A.

- **Unateness of Timing Arc**

  When performing delay annotation from an input pin to an output pins of a gate, it is important to consider the **unateness** of the examining gate. Unateness describes the relationship between the input transition type and the result output transition type along a timing arc. This relationship is crucial for correctly computing the rise and fall arrival times and slews at the output.

- For a **positive unate** timing arc, a rising (falling) input transition causes a rising (falling) output transition.
- For a **negative unate** timing arc, the transitions are inverted: a rising (falling) input transition produces a falling (rising) output transition. Therefore, the rise characteristics at the input affect the fall characteristics at the output, and vice versa.
- For **non-unate** (binate) arcs, the output transition depends on multiple inputs and can't be strictly considered as positive or negative unate. Therefore, the rise characteristics at the input affect both the rise and fall characteristics at the output, in case the side inputs are not set to a logic value affecting the unateness of the binate arc.

Understanding unateness allows STA tools and your delay annotation algorithms to correctly apply the appropriate delay values and compute accurate rise and fall delays and slews. For example, in a negative unate arc, if the input signal's rise arrival time is known, it should be used to compute the corresponding fall arrival time at the output, reflecting the inversion.

- **Gate-to-Gate Connections and Interconnect Delay**

  Your annotation should focus solely on gate delays, **ignoring the interconnect (wire) delays** in this assignment. This assumption simplifies the problem by equating the delay at a gate's input pins with the output pin delay of its driving gate(s).

- **Handling Liberty "When" Conditions**

  Liberty (.lib) files frequently specify timing arcs whose delay apply only under specific logical conditions, denoted as **"when" conditions**. Correctly evaluating these conditions is essential for accurate delay annotation, particularly in designs where some gate pins may be set to constant values through logic or synthesis.

  In practice, it is needed to propagate constant signals throughout the netlist in order to determine which pins are affected, allowing for precise selection of relevant timing arcs. For instance, on a 2-input OR gate, tying one input to `LOGIC-1` forces its output to `LOGIC-1`, since this is the controlling value. For the purposes of this assignment, all designs come with a precomputed constant propagation logfile, named as `<design>-constants.log`, listing each pin and its constant value, as below:

```
M1_inst/M2_inst/U1/A 0
M1_inst/M2_inst/U1/Z 1
...
```

  For each cell, when some input pins have constant values and timing arcs include "when" conditions (e.g. `when: "(A1 & A2)"`), you are required to evaluate these Boolean expressions based on the current constance values of the input pins. This evaluation determines which timing arc and corresponding LUT should be selected for delay annotation.

  If the Boolean expression of a "when" condition evaluates to true given the propagated constants, select the timing arc and the associated LUT linked to that condition. If none of the "when" conditions are met, fall back to the default unconditional timing arc – typically representing the worst-case timing behaviour.

To perform the Boolean evaluation efficiently and correctly (respecting operator precedence), a stack-based evaluation method is recommended. You may consult the following reference for an example implementation [1].

- **Critical Path Extraction and Reporting**

  After forward topological traversal is completed and arrival times are computed for all pins, perform a backward levelised traversal starting from the output pin with the maximum arrival time (critical output). Using the previously computed logical levels, trace back through the sequence of gates responsible for the worst-case delay path by selecting, at each gate, the input pins and input-to-output arcs contributing to the maximum delay (we recommend you to keep the worst arc to be easy to print the path without extra traversal and computation). This results in the critical timing path in the circuit. Report this critical path, listing the gates and pins involved, along with their delay annotations. This process provides essential insight into timing bottlenecks and facilitates targeted optimisation.

- **Modular and Extensible Design**

  Leverage the templated and modular structure established in previous assignments for delay annotation. Your solution should maintain flexibility by encapsulating delay computation separately from traversal mechanisms, similar to unit-delay computation. Use C++ template parameters for different interpolations and extrapolation strategies. Such modularity promotes code reuse, maintainability, and extensibility for future enhancements or different timing models or interpolation/extrapolation strategies.

[1] `https://algo.monster/liteproblems/1106`

4 Assignment Objectives

1. **TCL Commands for Initial Conditions**

   Implement TCL commands to specify the input transition time and delay for each primary input, and the output load for each primary output.

2. **Implementation of Linear Interpolation and Extrapolation Methods**

   Develop and integrate linear interpolation and linear extrapolation algorithms to estimate delay values from Liberty LUTs.

3. **Liberty LUT-based Delay Annotation**

   Perform gate-level delay annotation by interpolating and extrapolating delay values from Liberty LUTs based on input slew and output load for each timing arc. Incorporate these delay computations seamlessly into the topological traversal of the digital circuit. During the delay annotation, when considering an input-to-output timing arc, compute the output delay/slew considering the arc unateness as well as the related pin of this arc in order to consider the proper input slew and delays.

   You need to rename the TCL command from the 2nd assignment for unit delay calculation to:

   calculate_delay <method>

   where `<method>` can be one of the following:

   - `unit_delay`: calculate logic levels and annotate unit delays
   - `LUT_delay`: calculate logic levels and perform LUT delay annotation

   For both methods, you should print the timing information for each pin based on the table below. For gatepins that have a constant value, the name of the gatepin and the level should be printed while for the `Rise Slew`, `Rise Arrival Time`, `Fall Slew`, and `Fall Arrival` a '*' should be printed. For the `unit_delay` method only, set `Rise Slew`, `Fall Slew`, and `Fall Arrival` to 0.

Gate/Pin	Level	Rise Slew	Rise Arrival Time	Fall Slew	Fall Arrival
...	...	...	...	...	...

   Table 1: Timing information to print for each pin.

4. **Critical Path Extraction and Reporting**

   At the end of the `LUT_delay` annotation, identify the critical timing path by tracing backwards from the output pins with the maximum arrival time, using level annotation to guide traversal. Report the critical path in the following format:

Gate/Pin	Level	Gate Delay	Arrival Time
P1	0	-	0
U1/A	1	-	0
U1/Z	2	1.0	1.0
U2/A	3	1.5	2.5
...	...	...	...

5 Deliverables

Students are required to submit the following for successful completion of this assignment:

1. **Code Changes**
   - Submit only the specific code changes made for this assignment using git.
   - The changes should include the implementation of the linear interpolation and extrapolation methods, the LUT-based delay annotation as well as the critical path extraction and reporting functionality.
   - Additional changes containing implementations for previous assignment should be included for evaluation.
   - Ensure that commits are well-documented with meaningful comments.
   - Include an automated script that loads all designs into the tool and, for each design, exports a separate file containing the critical path in the format described above.
2. **Presentation**
   - Prepare and deliver a presentation showcasing your implementation.
   - The presentation should include:
     - Explanation of the linear interpolation and extrapolation methodologies
     - Details on how LUT-based delay annotation is performed and integrated during the traversal
     - Description on the methodology for extraction of critical path of the design.
     - Discussion of design decisions and challenges faced.
     - Live demo running the tool for some of the provided designs.
     - Profiling data of your code about runtime and memory

6 Evaluation

- **Correctness of Delay Annotation and Critical Path Extraction**
  - The primary metric for evaluation is the accuracy of delay annotation for each gatepin, as well as the extracted critical path, in the provided test circuits.
  - A bonus will be awarded to those who attempt to optimise their data structures and algorithms for runtime and memory efficiency.
- **Code Quality and Structure**
  - Code must be modular, clean, and well-documented, demonstrating propser integration of level and delay annotation logic during topological traversal.
  - Efficient and effective use of C++ templates for delay annotation will be assessed based on design clarity and future extensibility.
- **Presentation and Communication**
  - The presentation you submit on eClass will also contribute to the overall evaluation of your work.

7 Acknowledgment

We would like to thank Synopsys® for providing us the software as well as the liberty parser.

A Non-Linear Delay Model (NLDM) Liberty Format and Lookup Tables (LUTs)

The Non-Linear Delay Model (NLDM) is a widely-used timing characterisation format within Liberty (.lib) library files for digital circuits. It models the delay and output transition times of standard cells as functions of two primary electrical parameters:

- **Input Transition Time (Input Slew)**: The time is takes for the input signal to change from low to high (rise) or high to low (fall).
- **Output Load Capacitance**: The capacitive load driven by the cell's output, which affects how quickly the output signal can transition.

The NLDM format captures the relationship between these parameters and timing through **lookup tables (LUTs)**. Each LUT consists of delay or transition values tabulated of discrete points for various input slews and output loads. These tables enable detailed modeling of cell timing behaviour across a range of electrical conditions, providing more accuracy than simple fixed-delay models.

In Liberty files, timing arcs, (i.e. paths from specific input pins to output pins) specify NLDM LUTs that define the delay associated with that arc. When a timing analysis query requests delay for a given slew and load combination, the LUT provides data for the closest tabulated points.

Since the actual input slew and output load often fall between the discrete points in the LUT, interpolation is used to estimate delay values within the known range smoothly. If the query is outside the LUT's known range, extrapolation is applied to estimate the delay conservatively beyond the existing data.

Here is a short example illustrating a standard-cell with 2D NLDM delay and transition LUTs, typical in Liberty format:

```
cell (INVERTER) {
  pin (A) {
    direction: input;
  }
  pin (Z) {
    direction: output;
    max_transition: 1.0;
    timing () {
      related_pin : "A";
      timing_sense : positive_unate;

      cell_rise(delay_template_3x3) {
        index_1 ("0.1, 0.3, 0.7");   // Input transition times (ns)
        index_2 ("0.2, 0.4, 0.6");   // Output load capacitance (pF)
        values (                     // Corresponding rise delay values (ns)
          "0.12 0.15 0.21",
          "0.20 0.22 0.30",
          "0.35 0.37 0.45"
        );
      }

      cell_fall(delay_template_3x3) {
        index_1 ("0.1, 0.3, 0.7");
        index_2 ("0.2, 0.4, 0.6");
        values (
          "0.10 0.13 0.18",
          "0.18 0.20 0.25",
          "0.30 0.33 0.40"
        );
      }

      rise_transition(delay_template_3x3) {
        index_1 ("0.1, 0.3, 0.7");
        index_2 ("0.2, 0.4, 0.6");
        values (
          "0.08 0.09 0.11",
          "0.12 0.13 0.15",
          "0.20 0.22 0.25"
        );
      }

      fall_transition(delay_template_3x3) {
        index_1 ("0.1, 0.3, 0.7");
        index_2 ("0.2, 0.4, 0.6");
        values (
          "0.07 0.08 0.10",
          "0.10 0.11 0.13",
          "0.18 0.20 0.23"
        );
      }
    }
  }
}
```

This example defines a simple inverter cell with input pin A and output pin Z. For the output pin Z, the cell contains timing arcs related to the input pin A. Four 3x3 lookup tables are provided for rise delay, fall delay, rise, transition, and fall transition times.

Each LUT is indexed by two parameters: input transition times (`index_1`) and output load capacitance (`index_2`). The values table contains the corresponding delay or transition times for each combination of these indexes.

During static timing analysis, delays for arbitrary input slew and output load values are interpolated from these tables to estimate accurate timing behaviour.

This format is representative of how NLDM-based timing information is stored in Liberty files for detailed gate-level static timing analysis.

A.1 How to Access the LUT of an Arc

We provide an API that allows you to retrieve that allows you to retrieve the `index1` (input slew), `index2` (output load), and the corresponding `values` for any timing arc and attribute you request.

Use the function:

```
void getLUTDataFromTimingArc(
    LibTimingArc *arc,
    LUTType type,
    long double **input_slews,
    int *slews_size,
    long double **output_loads,
    int *loads_size,
    long double **values
);
```

where:

- `arc`: the timing arc of interest
- `type`: one of `CELL_RISE`, `CELL_FALL`, `RISE_TRANSITION`, or `FALL_TRANSITION`
- `input_slews`: pointer returning the input-slew index array
- `slews_size`: size of the input-slew index array
- `output_loads`: pointer returning the output-load index array
- `loads_size`: size of the output-load index array
- `values`: pointer returning the flattened 2D LUT values

You can also use the TCL function `get_cell_LUT`. Usage:

```
get_cell_LUT <cell_name> <lut_type>
```

, where `lut_type` is one of: `cell_rise`, `cell_fall`, `rise_transition`, `fall_transition`. This command returns all LUTs associated with a specific library cell for the selected type, allowing you to inspect the complete set of timing tables for that cell.

B Linear Interpolation and Extrapolation

This section provides a step-by-step example on how to perform linear interpolation and extrapolation on NLDM LUTs.

```
cell_rise(delay_template_3x3) {
    index_1 ("0.1, 0.3, 0.7");   // Input transition times (ns)
    index_2 ("0.2, 0.4, 0.6");   // Output load capacitance (pF)
    values (                     // Corresponding rise delay values (ns)
      "0.12 0.15 0.21",
      "0.20 0.22 0.30",
      "0.35 0.37 0.45"
    );
}
```

B.1 Linear Interpolation Example

Query Point for Interpolation

- Input slew $x = 0.2ns$
- Output load $y = 0.3pF$ (between 0.2 and 0.4pF)

Linear Interpolation Steps

- **Step 1: Identify bounding points in LUT around (x, y)**

$$x_1 = 0.1, \quad x_2 = 0.3, \quad y_1 = 0.2, \quad y_2 = 0.4$$

Corresponding delay values:

$$f_{11} = 0.12, \quad f_{12} = 0.15, \quad f_{21} = 0.20, \quad f_{22} = 0.22$$

- **Step 2: Interpolate along input slew (x) dimension at fixed output load**

At $y_1 = 0.2pF$:

$$f(x, y_1) = f_{11} + \frac{x - x_1}{x_2 - x_1}(f_{21} - f_{11}) = 0.12 + \frac{0.2 - 0.1}{0.3 - 0.1}(0.20 - 0.12) = 0.152$$

At $y_2 = 0.4pF$:

$$f(x, y_2) = f_{12} + \frac{x - x_1}{x_2 - x_1}(f_{22} - f_{12}) = 0.15 + 0.5 \times (0.22 - 0.15) = 0.185$$

- **Interpolate along output load (y) dimension at fixed input slew**

$$f(x, y) = f(x, y_1) + \frac{y - y_1}{y_2 - y_1}(f(x, y_2) - f(x, y_1)) = 0.152 + \frac{0.3 - 0.2}{0.4 - 0.2}(0.185 - 0.152) = 0.1685$$

Result: interpolated delay $\approx$ 0.1685ns.

B.2 Linear Extrapolation Example

Query Point for Interpolation

- Input slew $x = 0.05ns$ (below LUT minimum 0.1ns)
- Output load $y = 0.2pF$

Linear Extrapolation Steps

- **STEP 1: Calculate slope between nearest two LUT slew points at** $y = 0.2$

$$\text{slope} = \frac{f_{21} - f_{11}}{x_2 - x_1} = \frac{0.20 - 0.12}{0.3 - 0.1} = 0.4$$

- **STEP 2: Extrapolate delay at** $x = 0.05$

$$f(0.05, y) = f_{11} - \text{slope} \times (x_1 - x) = 0.12 - 0.4 \times (0.1 - 0.05) = 0.12 - 0.02 = 0.10$$

Result: extrapolated delay 0.10ns.

This linear extrapolation predicts a delay slightly lower than f_{11}, preserving trend continuity.

C Step-by-Step Worst Case Delay Example for a 4-Input Gate

Consider a logic cell with inputs `B1, B2, A1, A2`, and output `X`. The Liberty file for this cell defines multiple timing arcs from each input to the output pin `X`, some having "when" conditions and some unconditional (general) arcs.

Suppose constant input propagation gives:

- $A1 = 1'b0$
- $A2 = 1'b0$

C.1 Case 1: Arc `B1`→`X`

For `B1`→`X`, check available LUT/timing arc:

- "when: (A1 * A2)" evaluates to $(0 * 0) = 0$
- "when: (!A1 * A2)" evaluates to $(1 * 0) = 0$
- "when: (A1 * !A2)" evaluates to $(0 * 1) = 0$
- **"when: (!A1 * !A2)" evaluates to (1 * 1) = 1 (condition is met)**

Since the "when: (!A1 * !A2)" condition is satisfied, choose the LUT and timing data associated with this specific condition to annotate the delay from `B1` to `X`. You should interpolate using this LUT, as described in the previous section. Suppose that: $D(B1 \rightarrow X) = 0.5ns$.

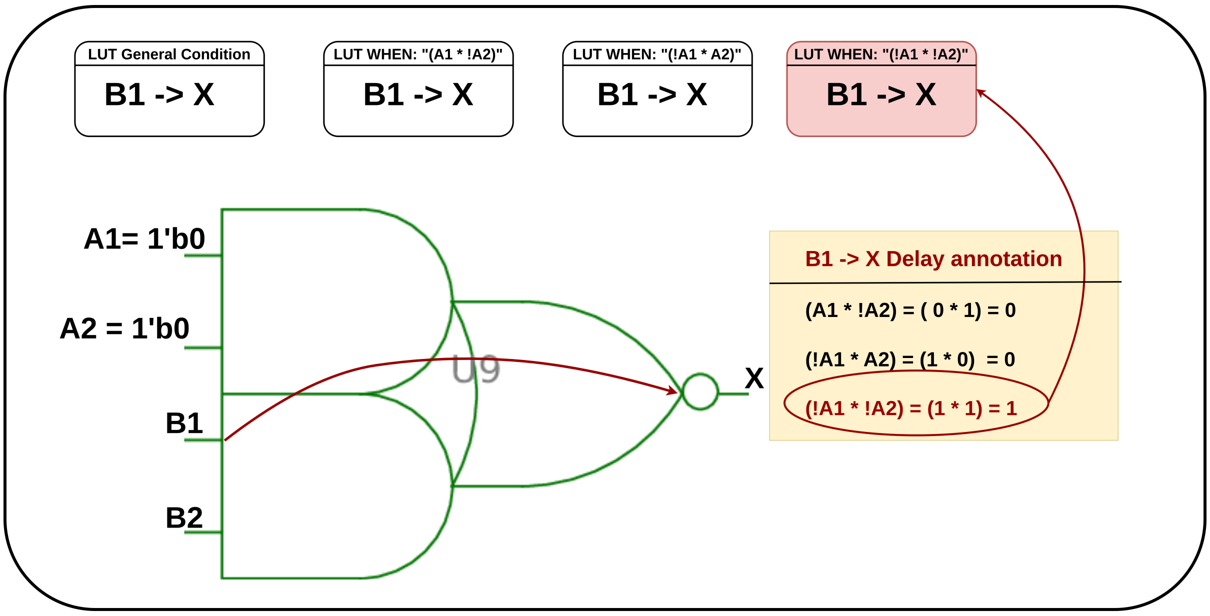

Figure 1: Handling **when** Conditions When the Condition Is **Met**

C.2 Case 2: Arc `B2`$\rightarrow$`X`

For `B2`$\rightarrow$`X`, check all "when" conditions:

- "when: (A1 * A2)" = (0 * 0) = 0
- "when: (!A1 * A2)" = (1 * 0) = 0
- "when: (A1 * !A2)" = (0 * 1) = 0

None of the "when" conditions are satisfied. Therefore, consider the general (unconditional) LUT for `B2`$\rightarrow$`X`, using this LUT to annotate the delay from `B2` to `X`. You should interpolate using this LUT, as described in the previous section. Suppose that: $D(B2 \rightarrow X) = 0.8ns$

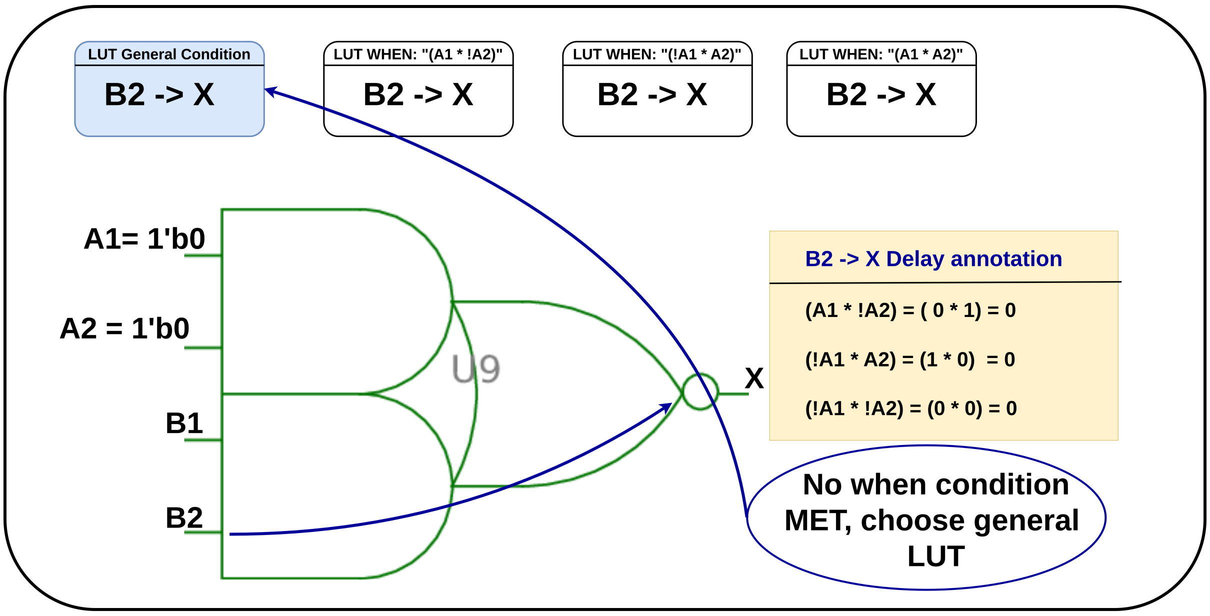

Figure 2: Handling **when** Conditions When the Condition Is **not Met**

C.3 Delay Annotation at `X`

After the "when" conditions evaluation process is completed both for `B1`$\rightarrow$`X` and `B2`$\rightarrow$`X` delay annotation at `X` needs to be performed considering the appropriate LUTs.

Given:

- Arrival times (AT) at inputs:
  - $AT(B1) = 2.1ns$
  - $AT(B2) = 2.0ns$
- Liberty LUT delay (D) values (interpolated) for the input-to-output arcs:
  - $D(B1 \rightarrow X) = 0.5ns$
  - $D(B2 \rightarrow X) = 0.8ns$

Step 1: Calculate AT at X via $B1$

- $AT(B1 \rightarrow X) = AT(B1) + D(B1 \rightarrow X) = 2.1 + 0.5 = 2.6ns$

Step 2: Calculate AT at X via $B2$

- $AT(B2 \rightarrow X) = AT(B2) + D(B2 \rightarrow X) = 2.0 + 0.8 = 2.8ns$

Step 3: Select worst case (maximum) arrival time (AT) for X

- $AT(X) = max\{AT(B1 \rightarrow X), AT(B2 \rightarrow X)\} = max\{2.6, 2.8\} = 2.8ns$

Although `B2` arrives earlier, the longer LUT delay for the `B2`→`X` arc results in a higher total delay (2.8ns). Input `B1` arrives slightly later but has a shorter LUT delay (2.6ns total). STA uses the worst case delay among all input paths (2.8ns) as the gate output delay, ensuring timing safety.