FMCAD 2022

Timed Causal Fanin Analysis for Symbolic Circuit Simulation

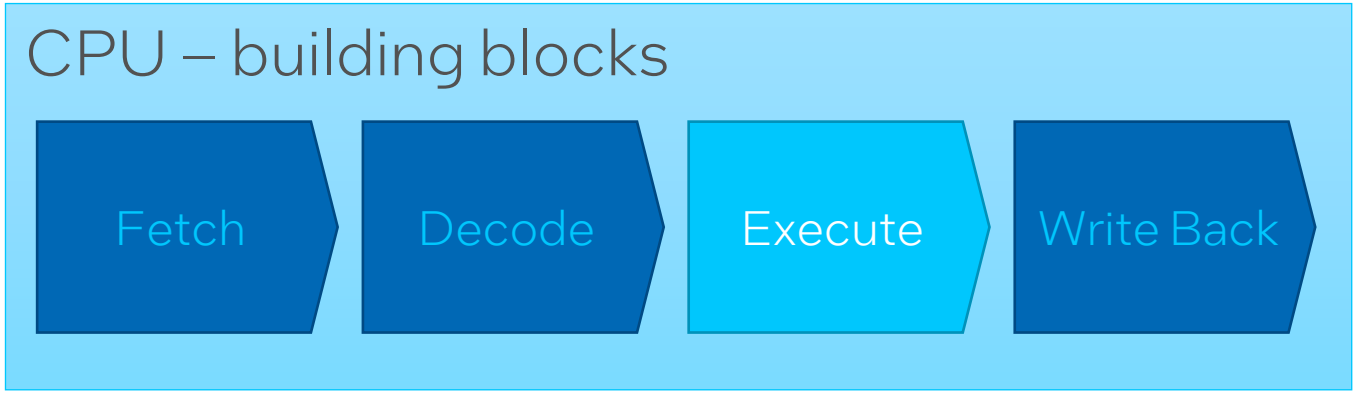
Roope Kaivola, Neta Bar-Kama

Core and Client Development Group, Intel

Agenda

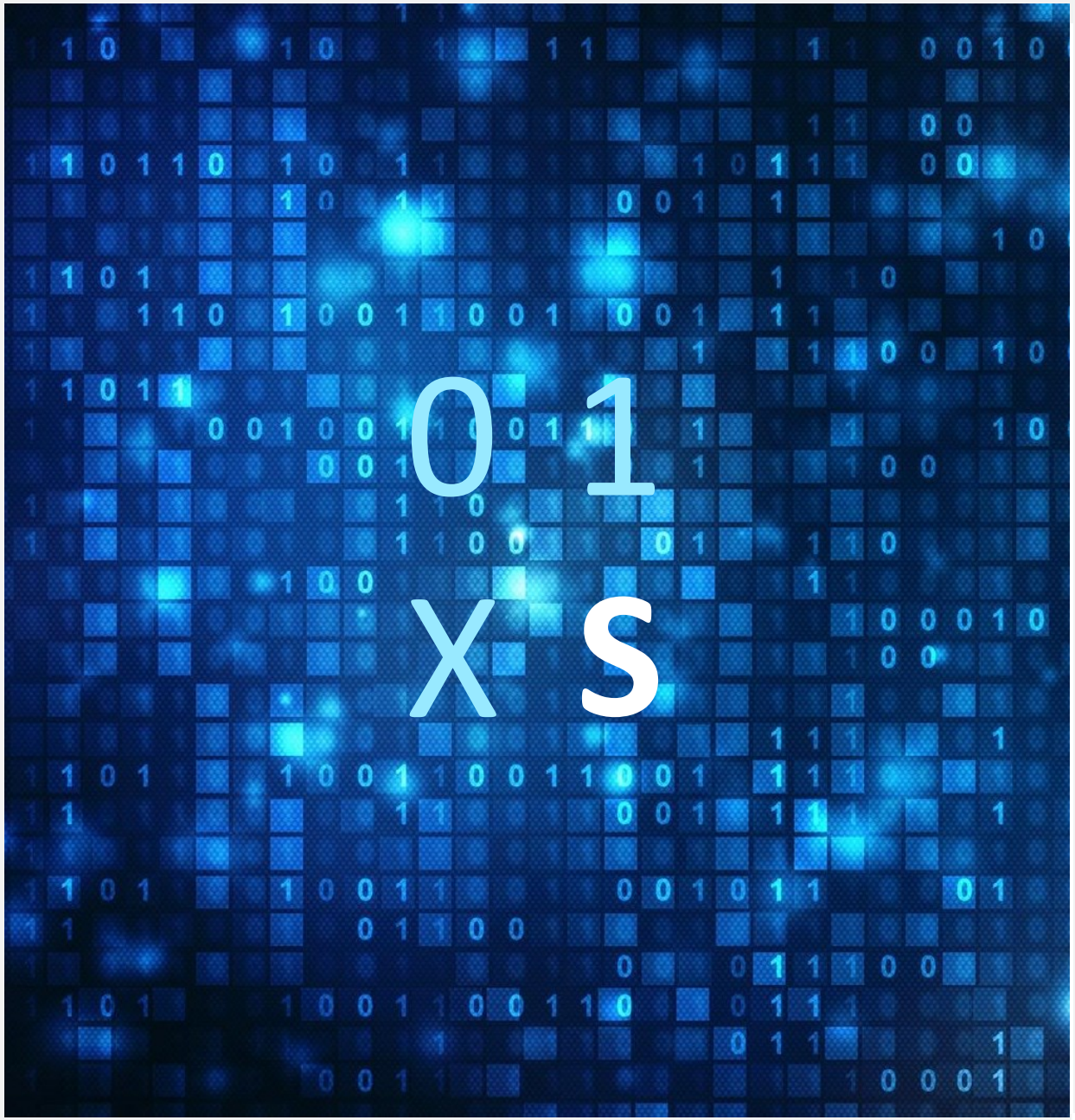
- Datapath formal verification
- Symbolic simulation
- Timed causal fanin analysis
- Results

Execution Cluster



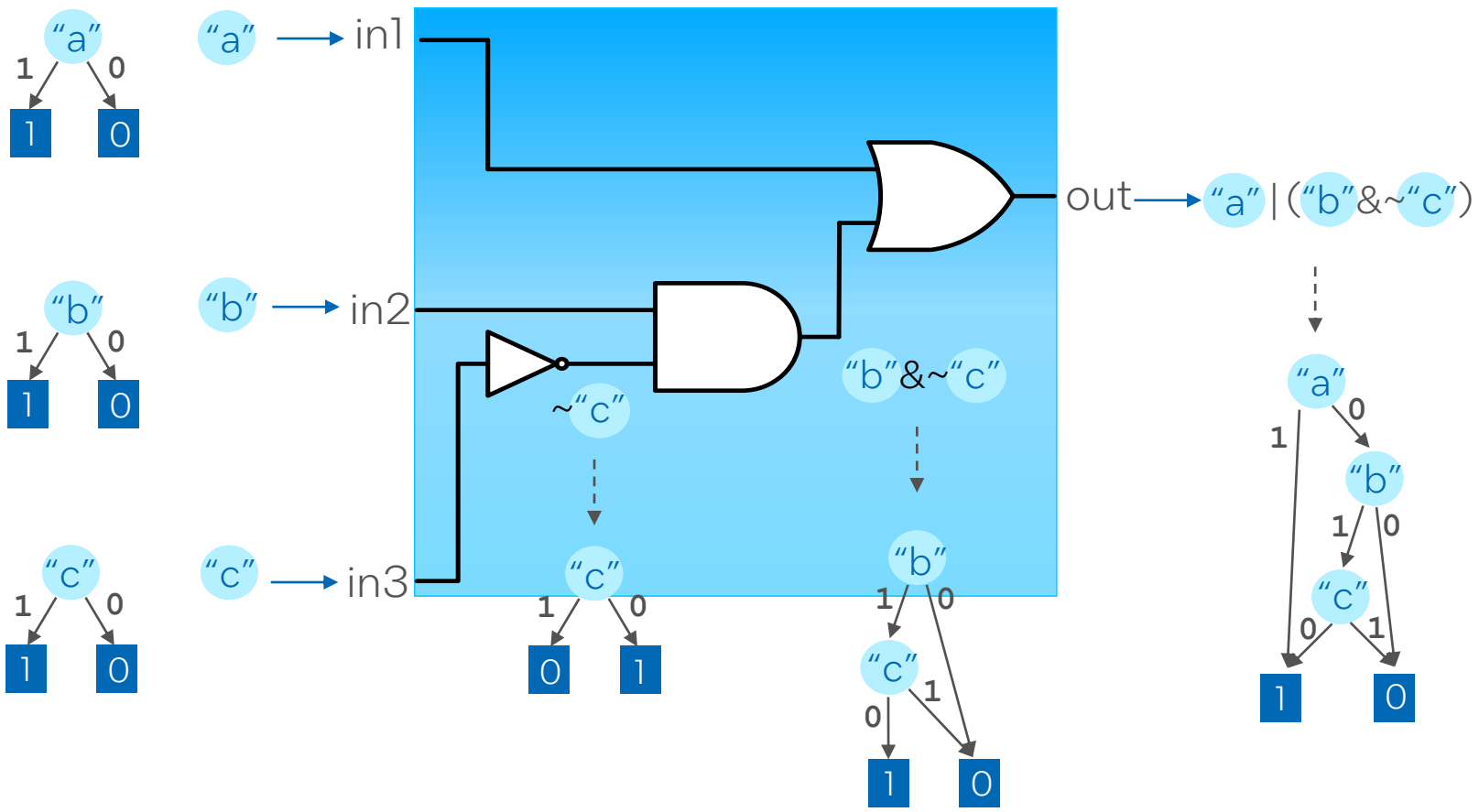
- The execution cluster (EXE) is a pipelined machine that receives streams of micro-operations (μops).
- ~5000 μops in Intel Core Processor EXE cluster.
  - Arithmetic, logic, branch operations, address calculations and more.
- Data calculations are performed on input sources and result goes to the write-back output.
- All EXE μops, especially floating-point complex computations, are formally verified using Symbolic Simulation (STE).

Symbolic Simulation



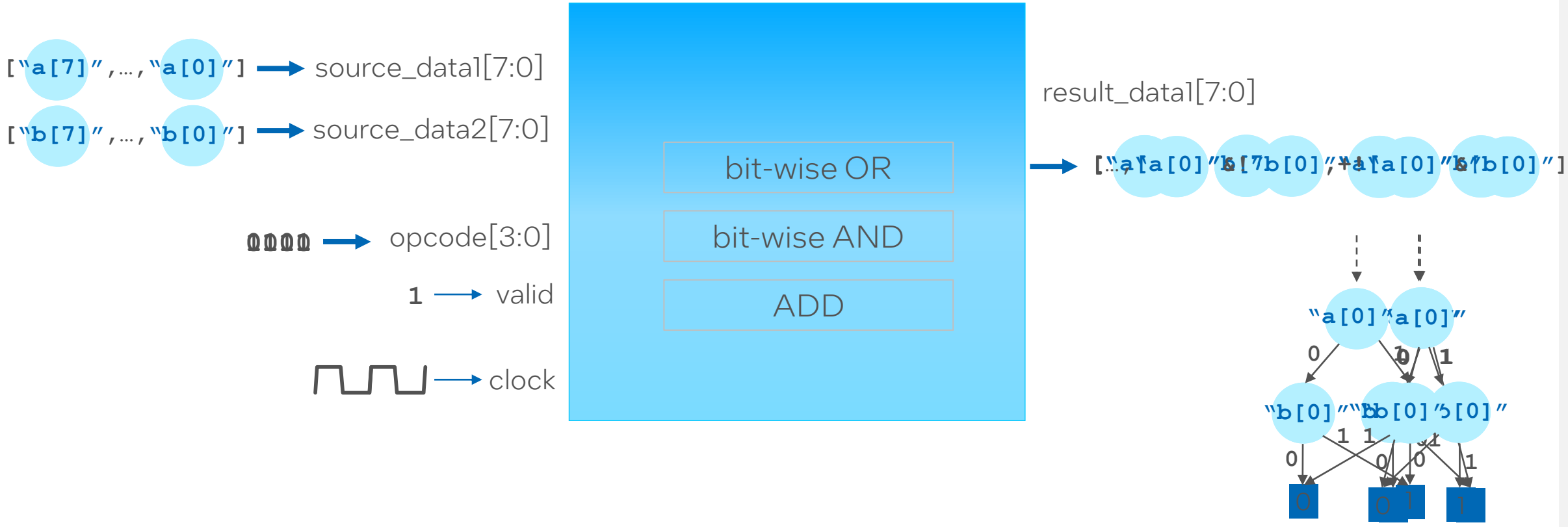
Symbolic Simulation

C. H. Seger and R. E. Bryant, "Formal verification by symbolic evaluation of partially-ordered trajectories," Formal Methods Syst. Des., vol. 6, no. 2, pp. 147–189, 1995



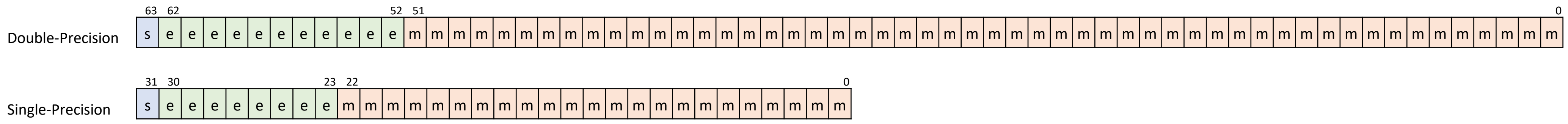
Symbolic Simulation

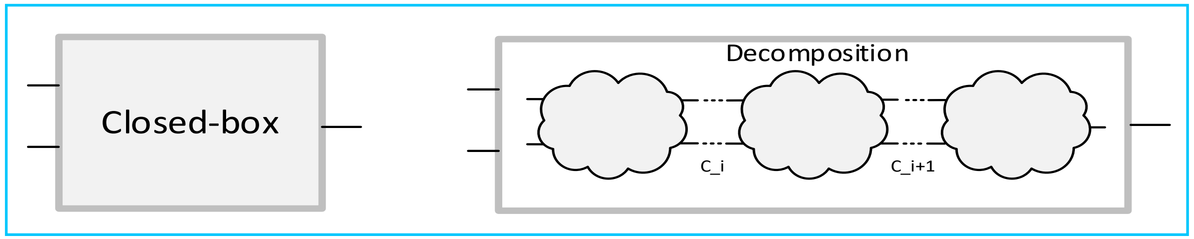
In 'real' life



Traditional Floating-Point Verification with Decomposition

- Floating point numbers: $$f = (-1)^{s} \times (1.m) \times 2^{(e-bias)}$$
- Traditional floating-point types: single, double and extended precision floats.





Decomposition

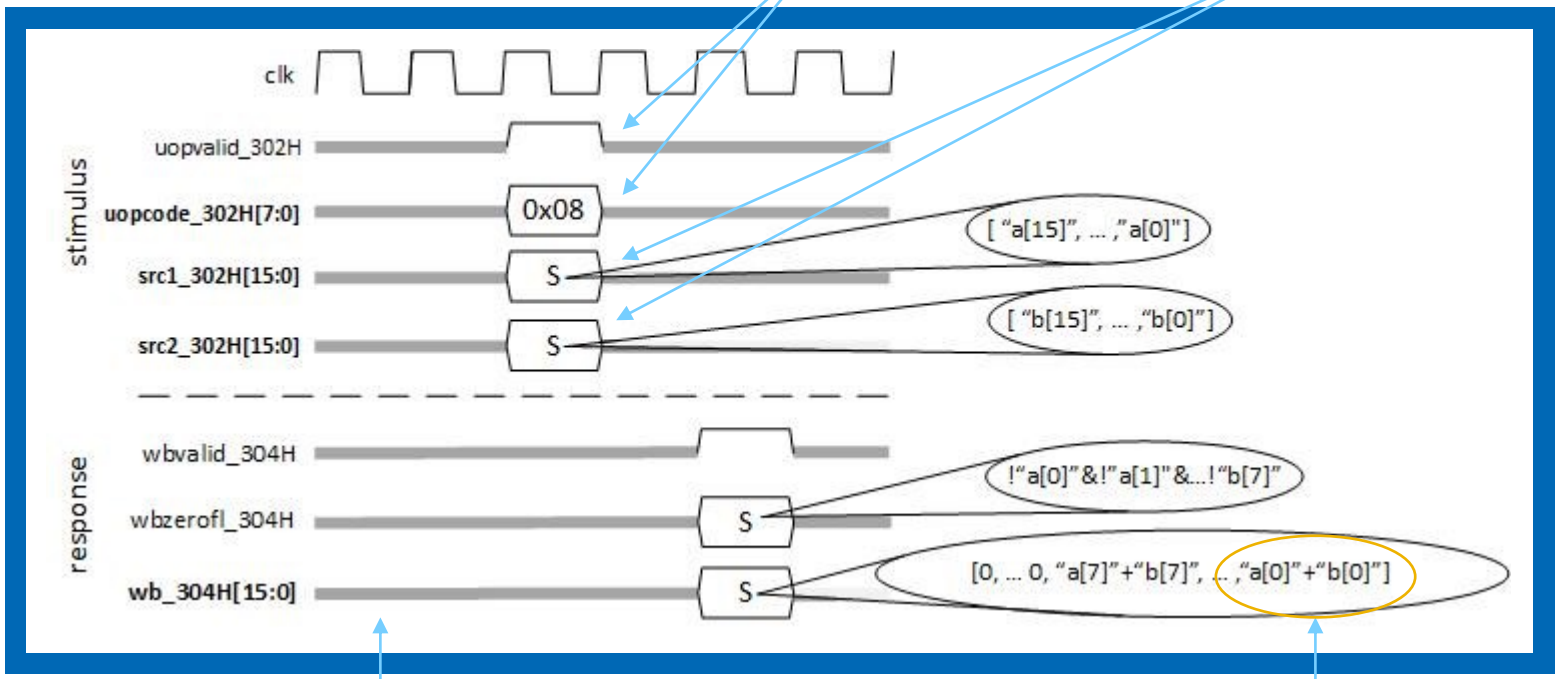
- Requires deep understanding of design implementation
- Cut-points side-conditions are not intuitive and hard to find
- Requires technical expertise

Symbolic Simulation

Waveform example

Constants driven to control signals

Symbols driven to data signals



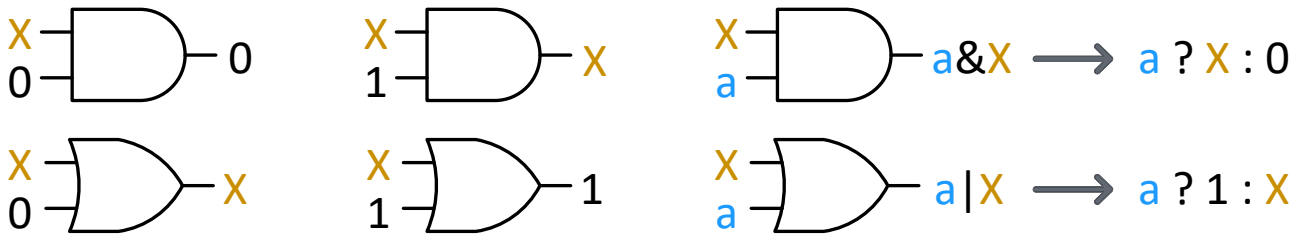
Every bit is a Boolean expression

Grey areas represent X – uninitialized/unknown values

Symbolic expressions sampled at the outputs

Calculations with ‘X’

Like a symbol, an X can also be 0 or 1 at any given moment, but it is not represented by a Boolean expression and doesn’t take memory.
In our terminology that is an **uninitialized** value, which is an **over-approximation** of the circuit behavior.



* **Weakening:**
Limiting the sets of signals, times or values in simulation, by replacing them with the undefined values ‘X’

Weakening

User-defined methods

- Universal
  - Replace with ‘X’ at all times
- Cycle-by-cycle
  - Replace with ‘X’ at specific times
- Dynamic
  - Replace with ‘X’ if the size of the expression exceeds a given threshold

* **Cycle-by-cycle** and **dynamic** weakening are unique to symbolic simulation

Parametric Substitution

Background and Motivation

- Symbolic simulation properties:

$$(trig_1 \wedge trig_2 \wedge \ \ldots \wedge trig_n) \rightarrow (goal_1 \wedge goal_2 \wedge \ \ldots \wedge goal_n)$$

- Simulate only when **all triggers are satisfied**
- Triggers simplify the simulation:
  1. Restrict simulation scope
  2. Reduce expression size

Parametric Substitution

Background and Motivation

Parametric substitution factors in triggers so that we can take advantage of them.

In ONE simulation, we cover ALL cases where triggers are satisfied, and ONLY those cases.

M. D. Aagaard, R. B. Jones, and C.-J. H. Seger, “Formal verification using parametric representations of Boolean constraints,” in DAC’99, pp. 402–407, 1999

R. B. Jones, Symbolic Simulation Methods for Industrial Formal Verification. Springer, 2002

Parametric Substitution

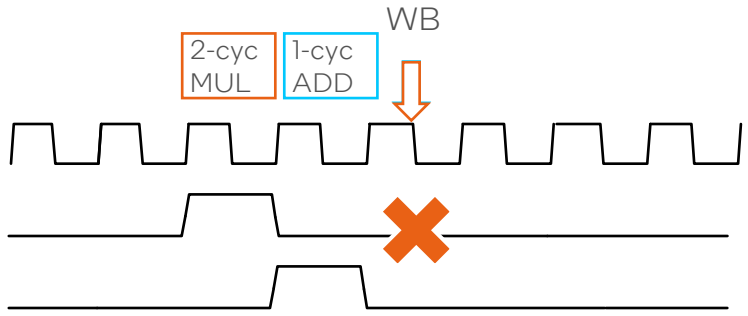
Single uop verification

Example:

We want to verify a 1-cyc latency ADD uop.

1. Basic condition for verification (constant values):
   uopcode = ADD
   valid = 1

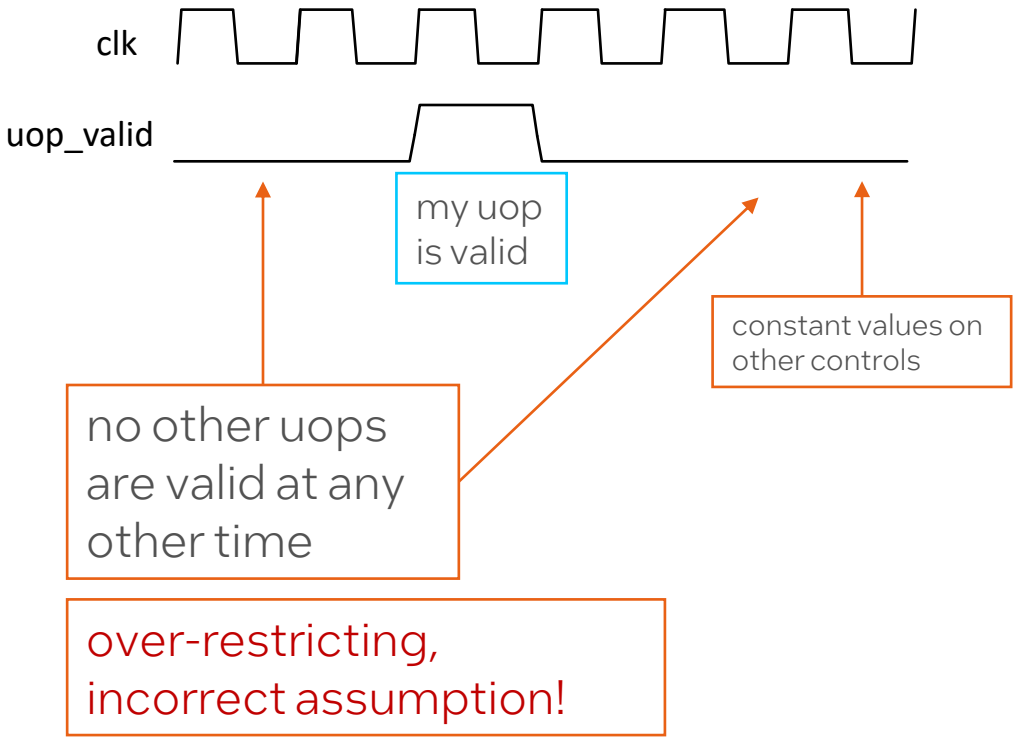
2. Prevent write-back clash:
   no other uop will write-back at the same time.



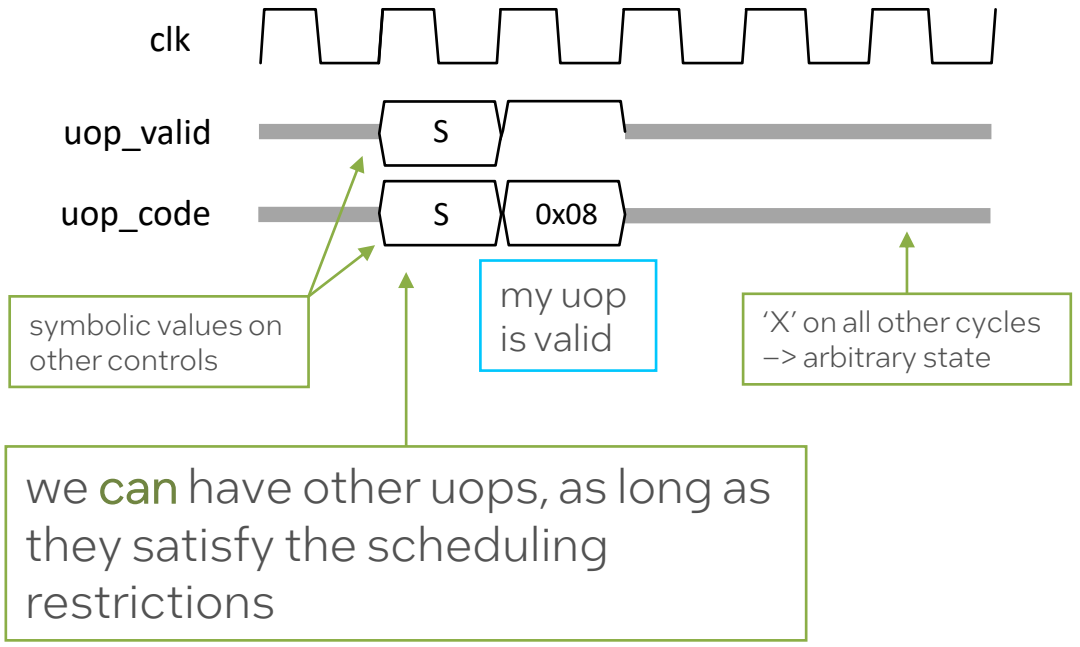
Parametric Substitution

Single uop verification

Option 1:



Option 2:



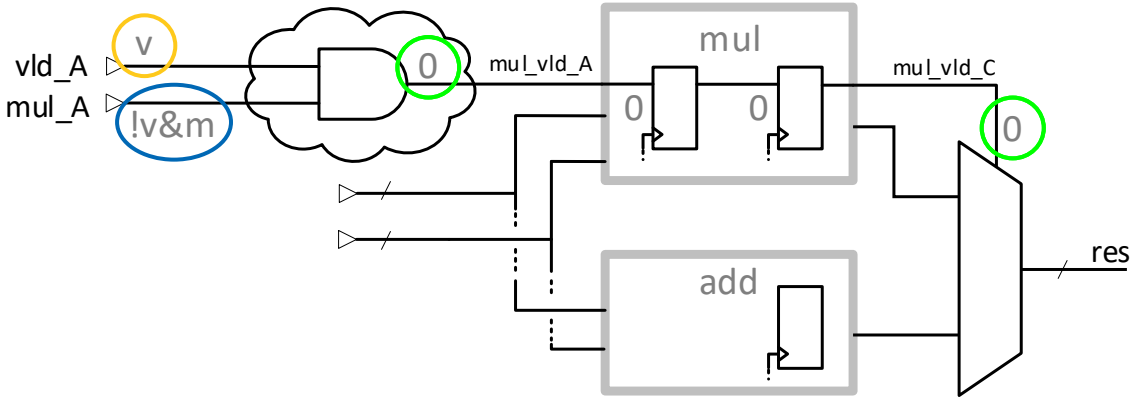
Parametric Substitution

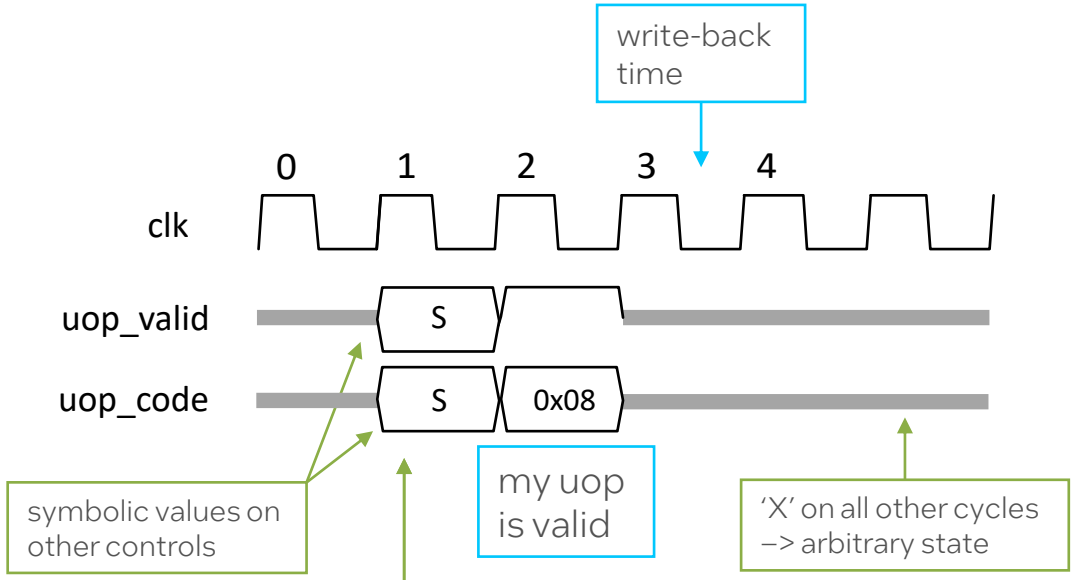
Single uop verification – internal constants derived from triggers

We cannot have two uops writing back at the same time
==
We cannot have:
a valid MUL 2-cyc uop at cycle 1
AND
a valid ADD 1-cyc uop at cycle 2

$$C_2(v,m) = \neg(v \wedge m)$$

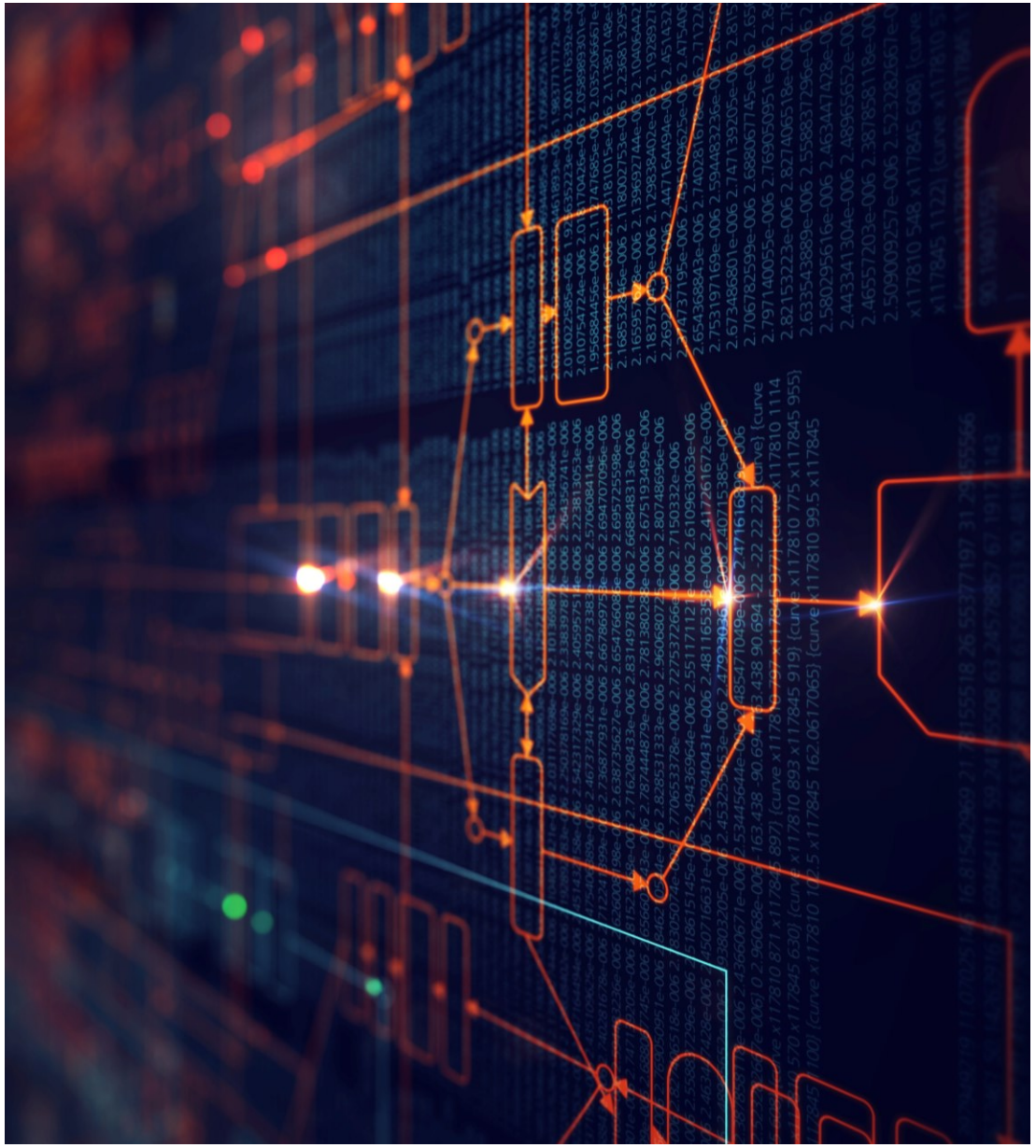
$$\vec{p} = \left[v \to v', m \to \left((\sim v') \wedge m'\right)\right]$$





we can have other uops, as long as they satisfy the scheduling restrictions

Timed Causal Fanin Analysis



Timed Causal Fanin Analysis

Data-path verification complexity reasons

1. Data-path complexity of the individual operation
2. Everything surrounding it

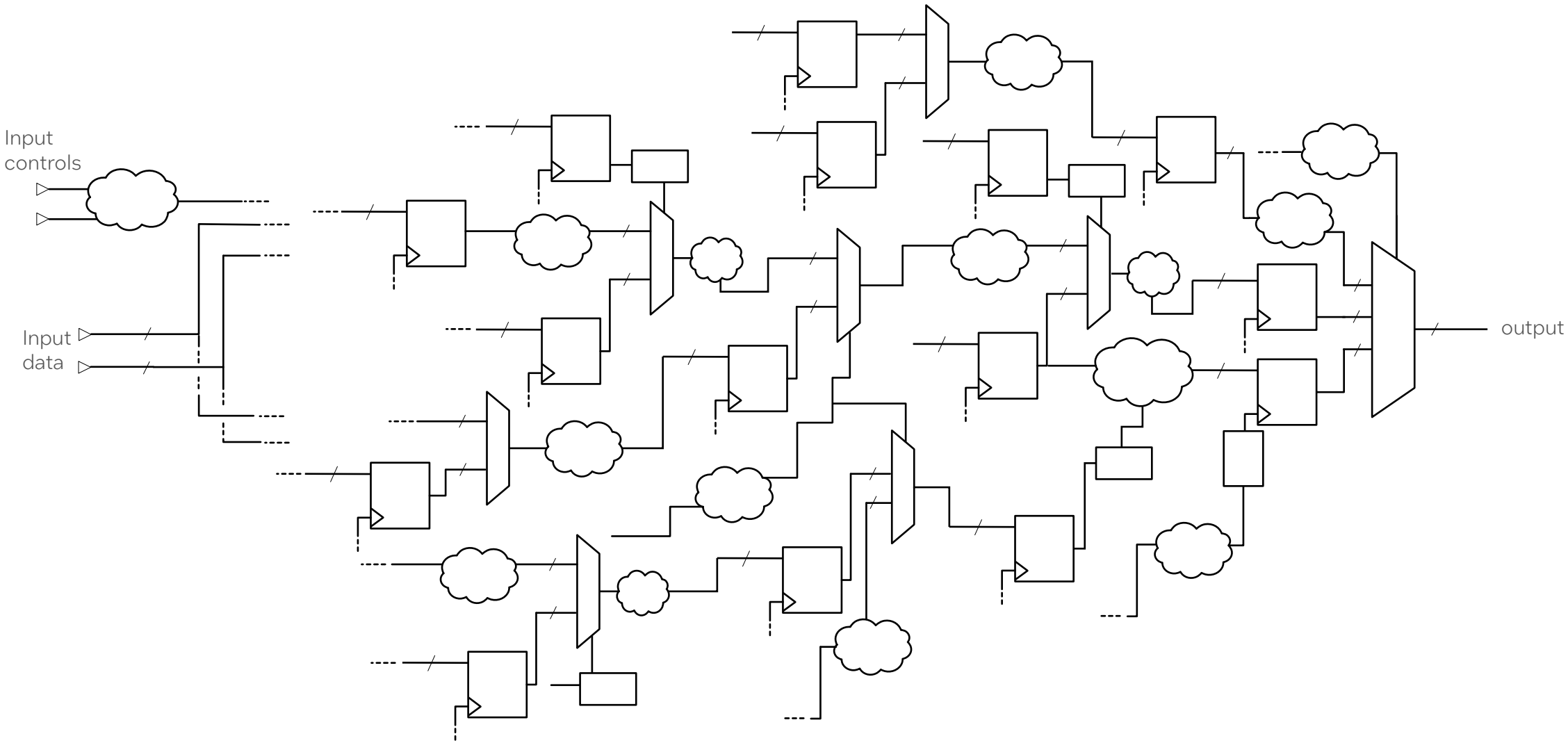
TCFA ingredients

- Cone of influence
- Constant based reduction
- **On a timed cycle-by-cycle basis**

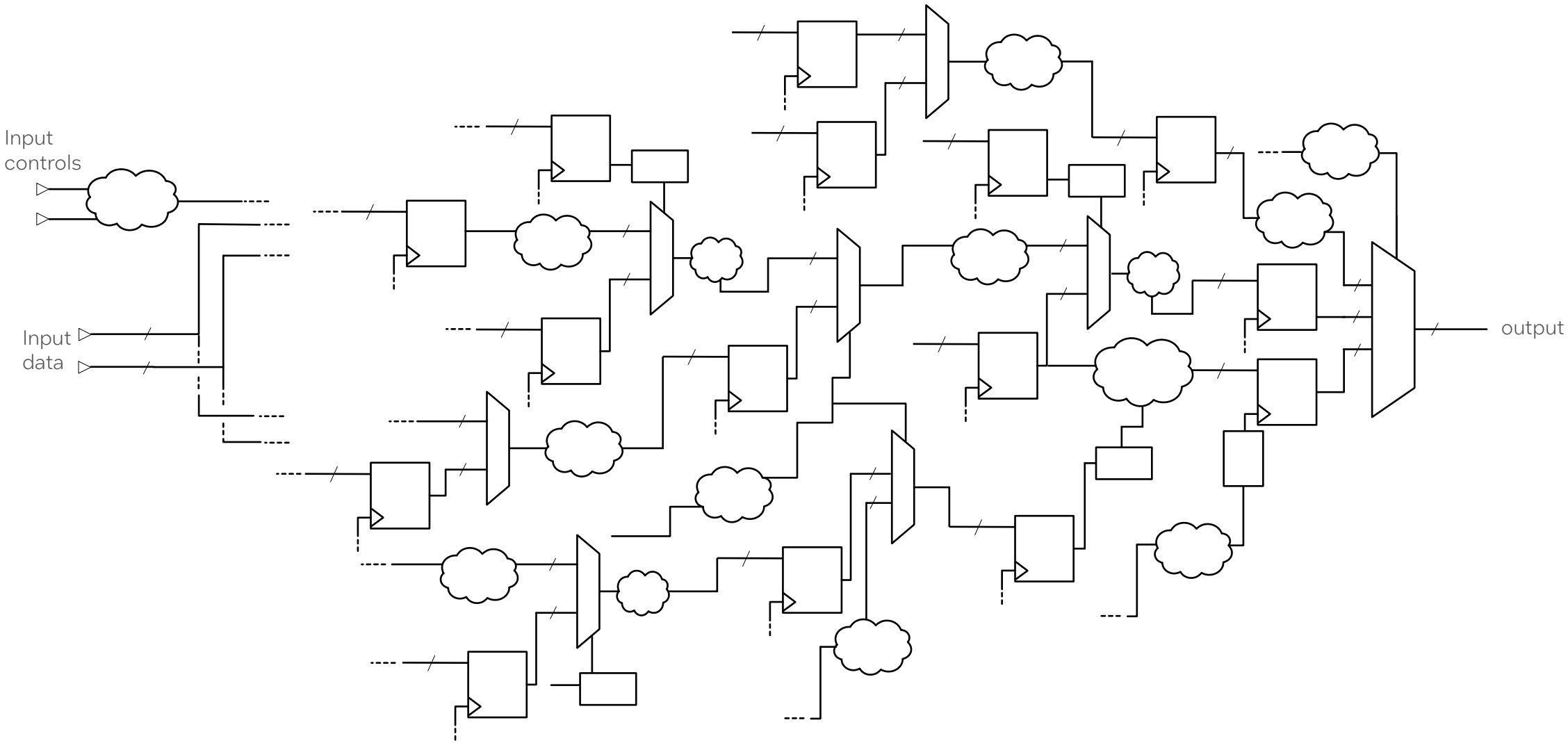
Underlying techniques

- Weakening
- Parametric substitution

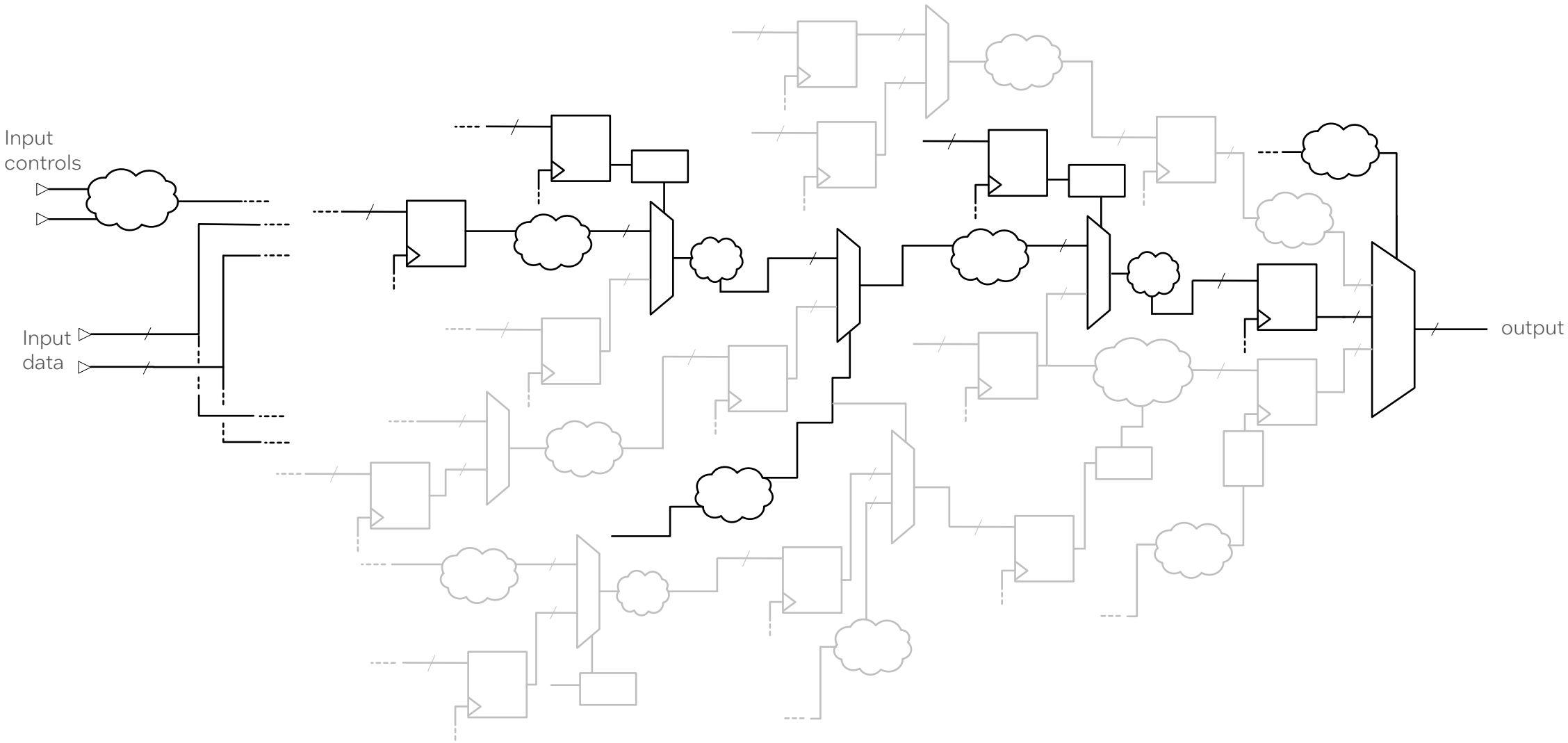
Timed Causal Fanin Analysis



Timed Causal Fanin Analysis



Timed Causal Fanin Analysis



Timed Causal Fanin Analysis

How does it work?

1. Initial simulation
2. Backwards traversal
3. Weakening list

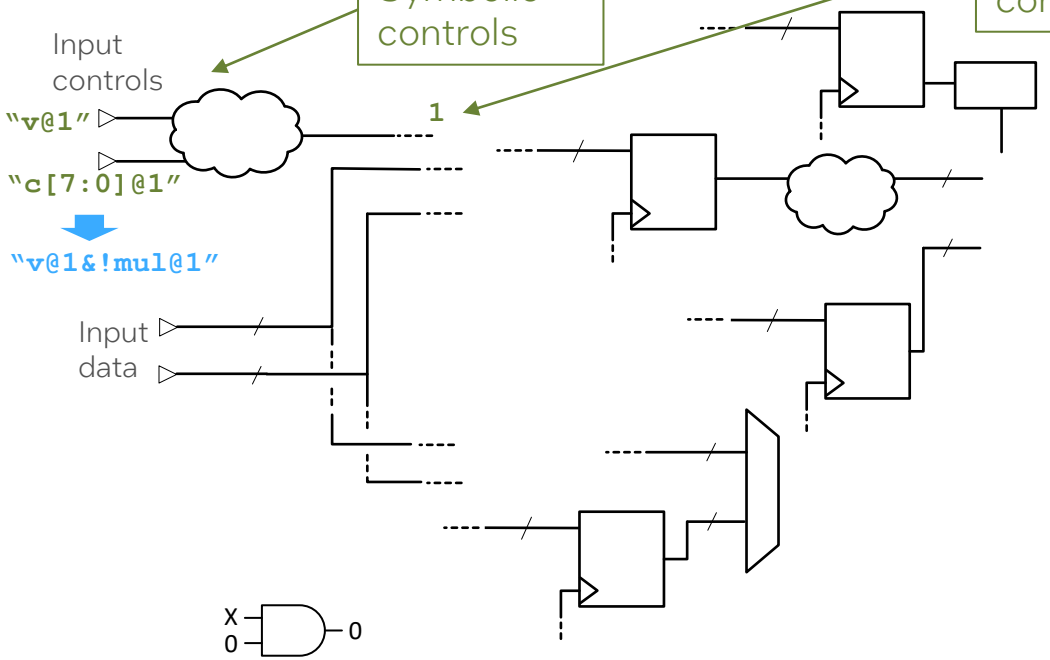
→ Main symbolic simulation

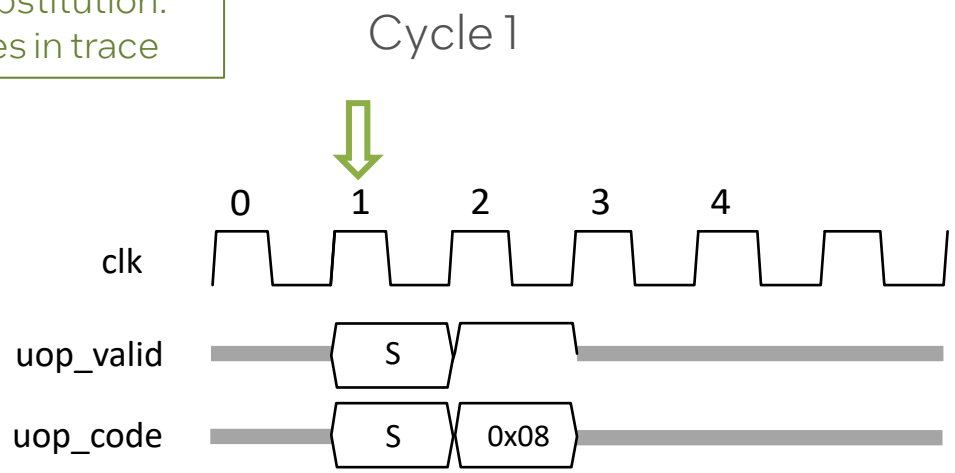
Timed Causal Fanin Analysis

Step 1: initial simulation

Symbolic controls

trigger combined with parametric substitution: constant values in trace

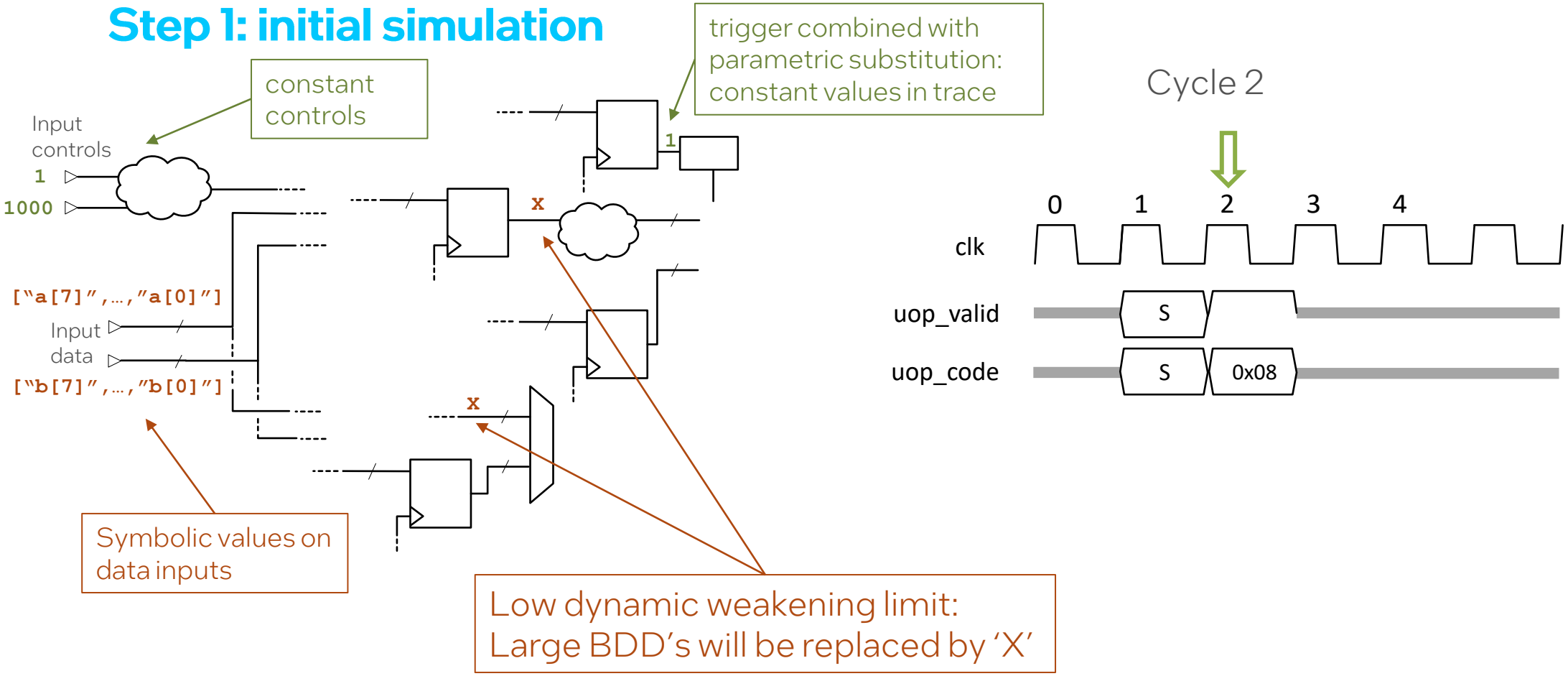




Other constants may be implied directly by circuit logic or trigger restrictions

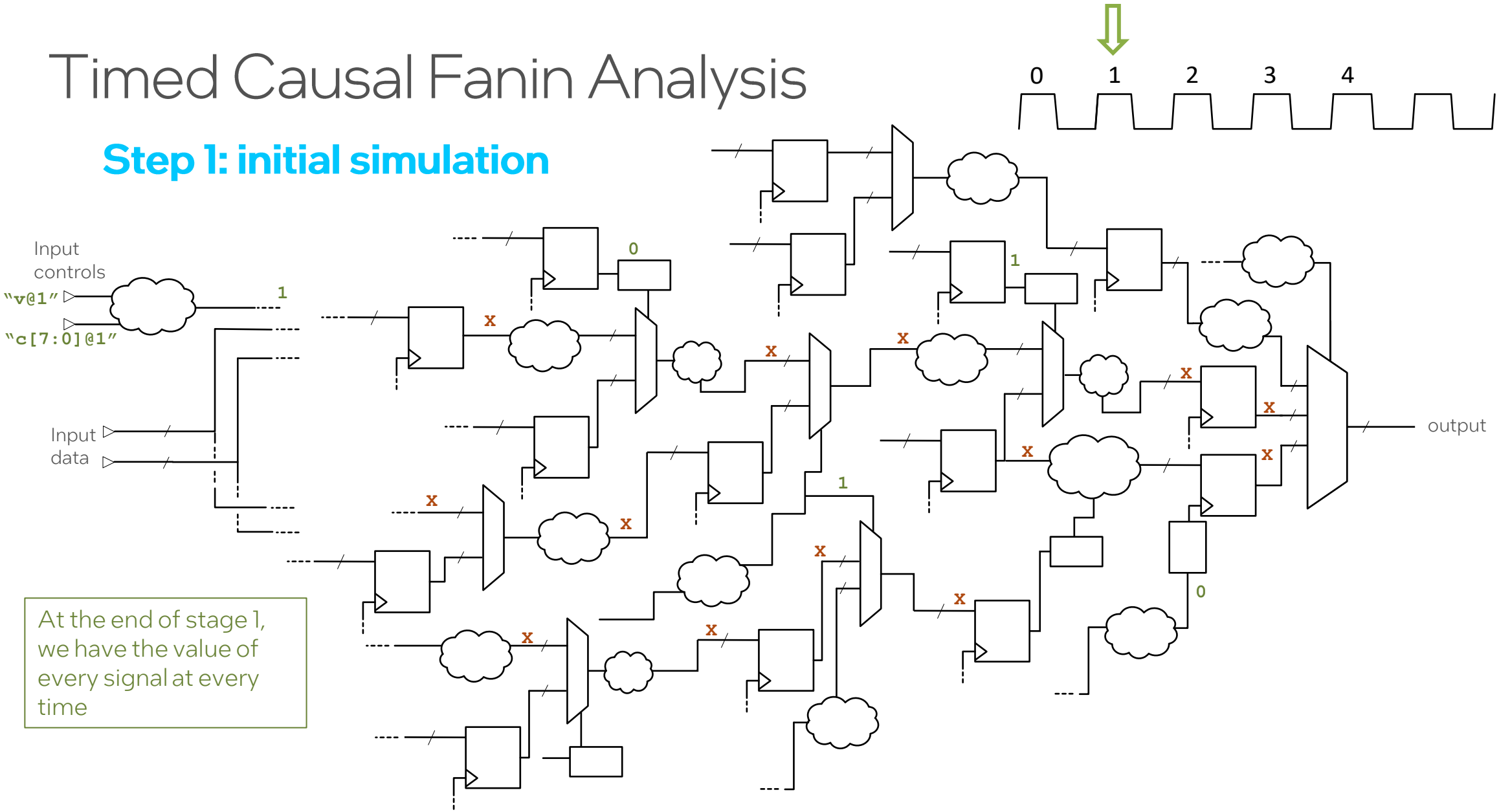
Timed Causal Fanin Analysis

Step 1: initial simulation



Timed Causal Fanin Analysis

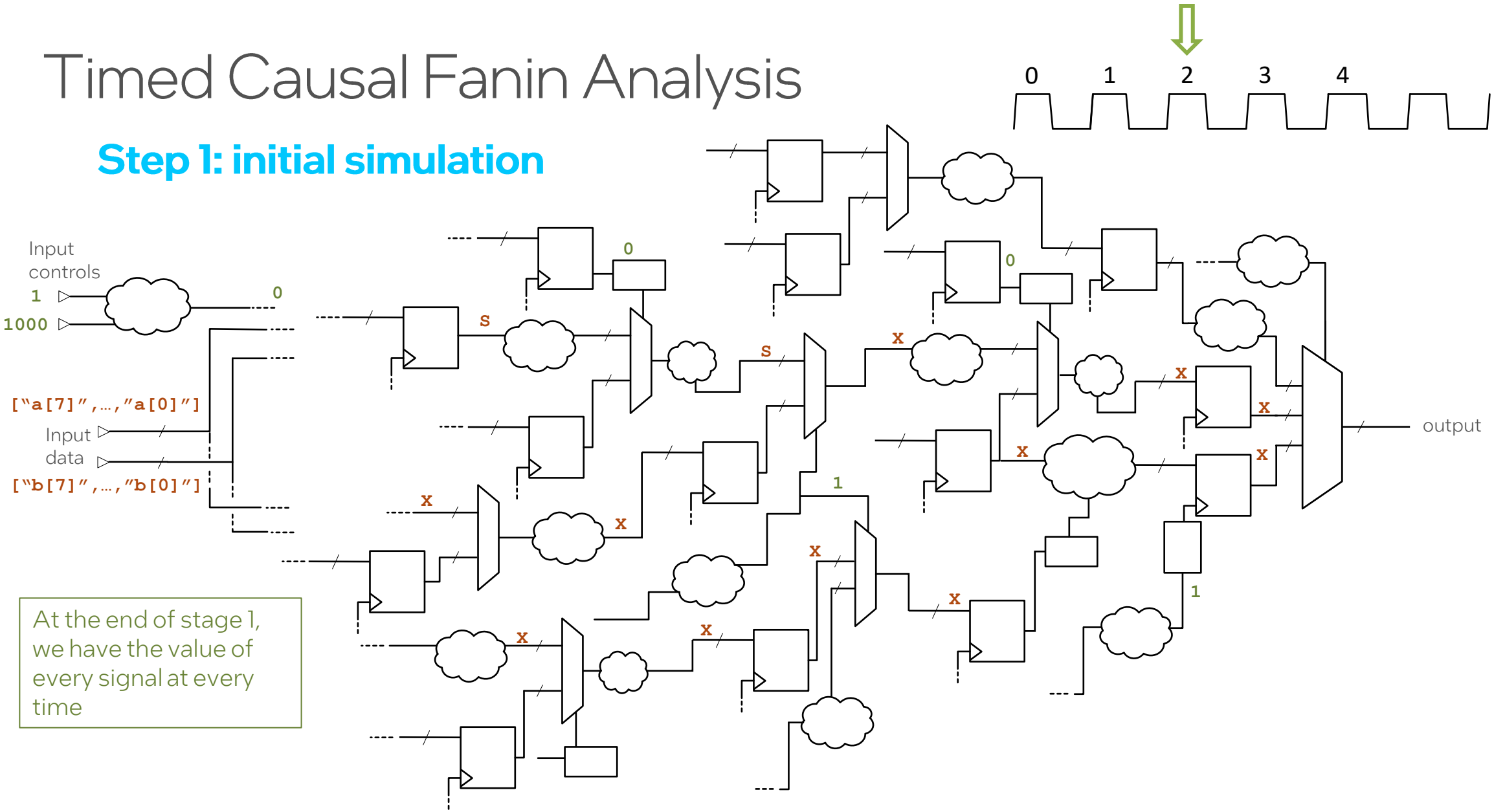
Step 1: initial simulation



At the end of stage 1, we have the value of every signal at every time

Timed Causal Fanin Analysis

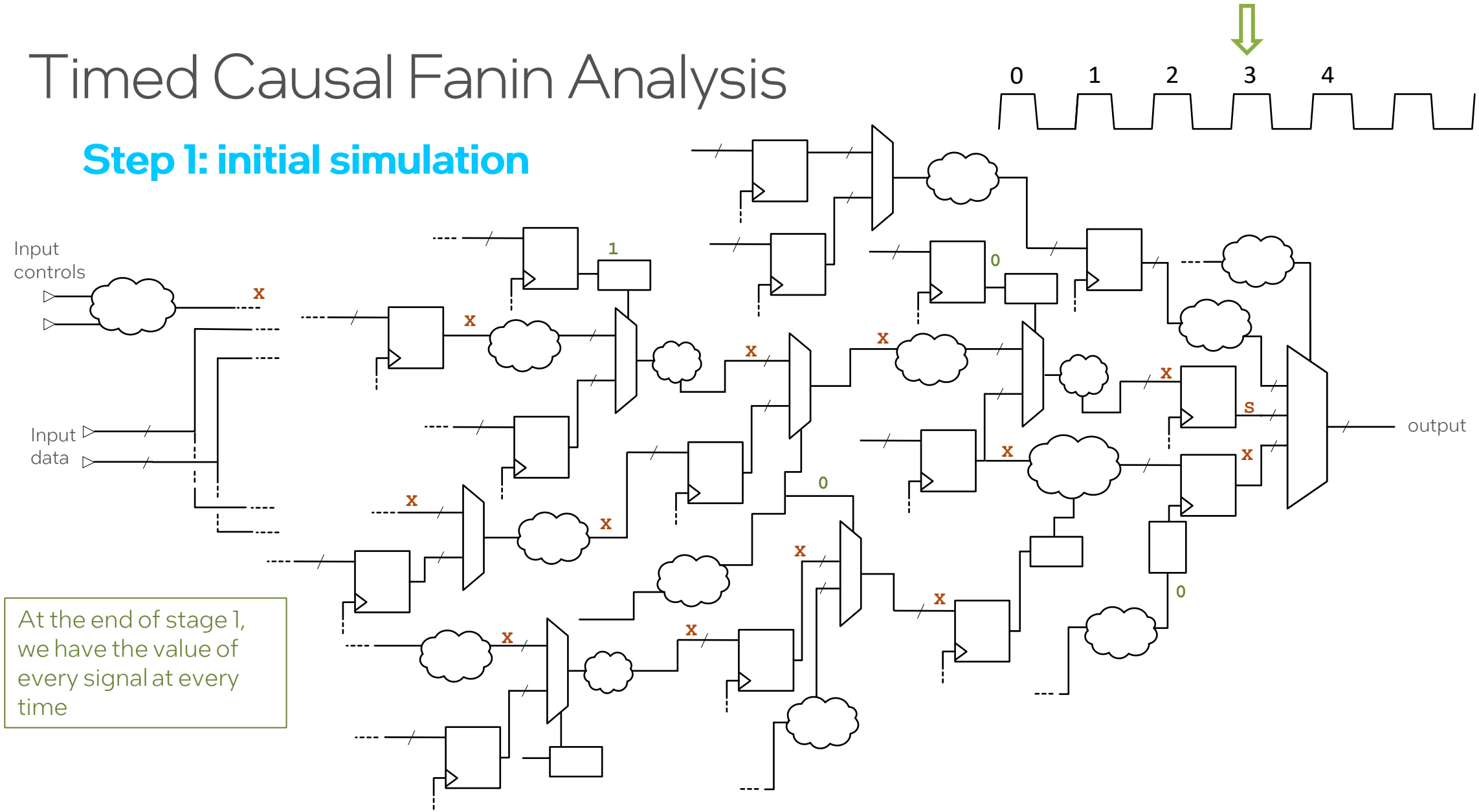
Step 1: initial simulation



At the end of stage 1, we have the value of every signal at every time

Timed Causal Fanin Analysis

Step 1: initial simulation

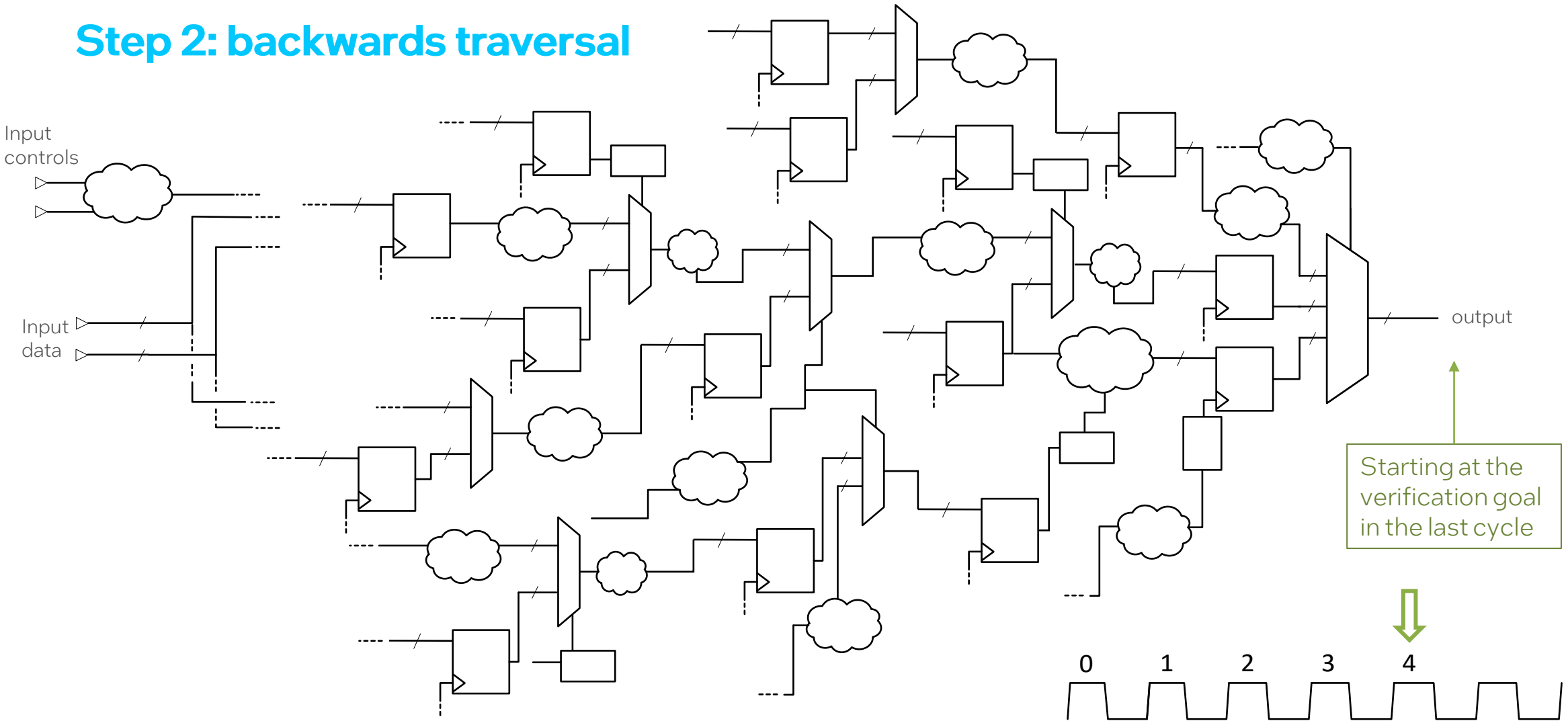


At the end of stage 1, we have the value of every signal at every time

Timed Causal Fanin Analysis

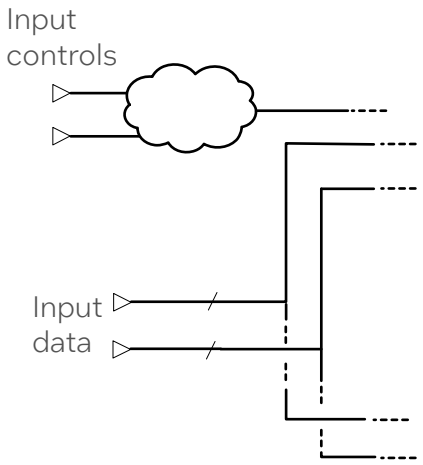
```
Causal fanin:
[output]
```

Step 2: backwards traversal



Timed Causal Fanin Analysis

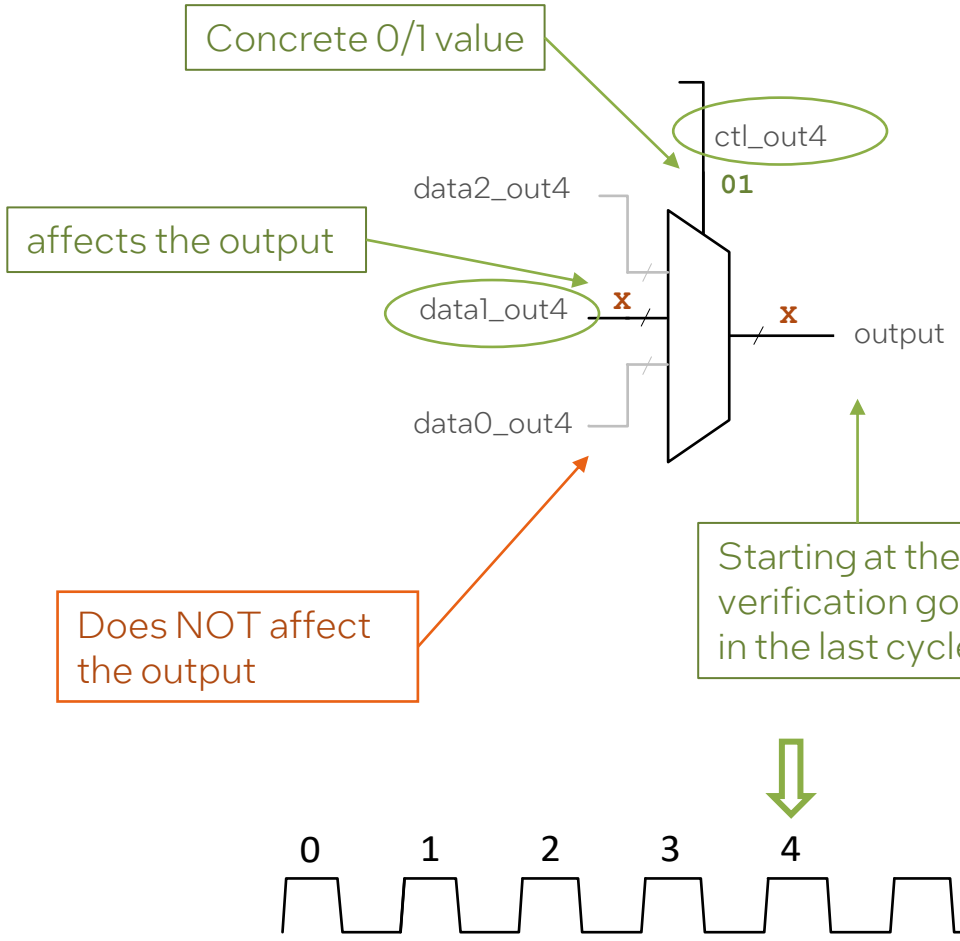
Step 2: backwards traversal




```
Causal fanin:
[output]
ctl_out4]
data1_out4]
```

Combinational gate

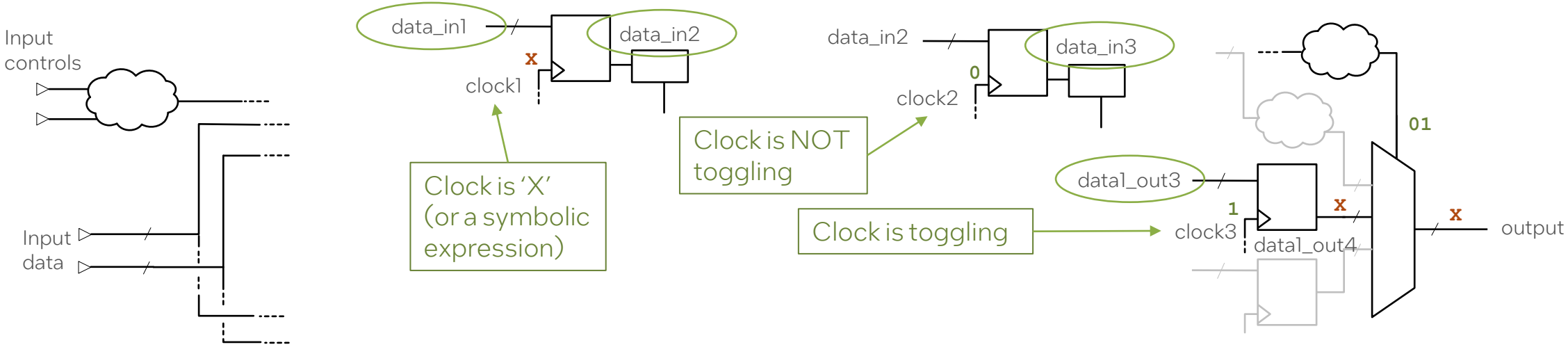
Concrete 0/1 value

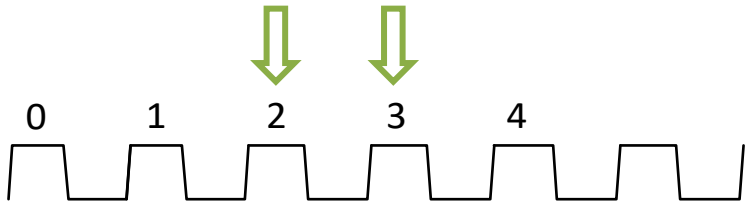


Timed Causal Fanin Analysis

Step 2: backwards traversal

```
Causal fanin:
[output,
ctl_out4,
data1_out4]
data1_out3]
data_in3]
data_in2,data_in1]
```

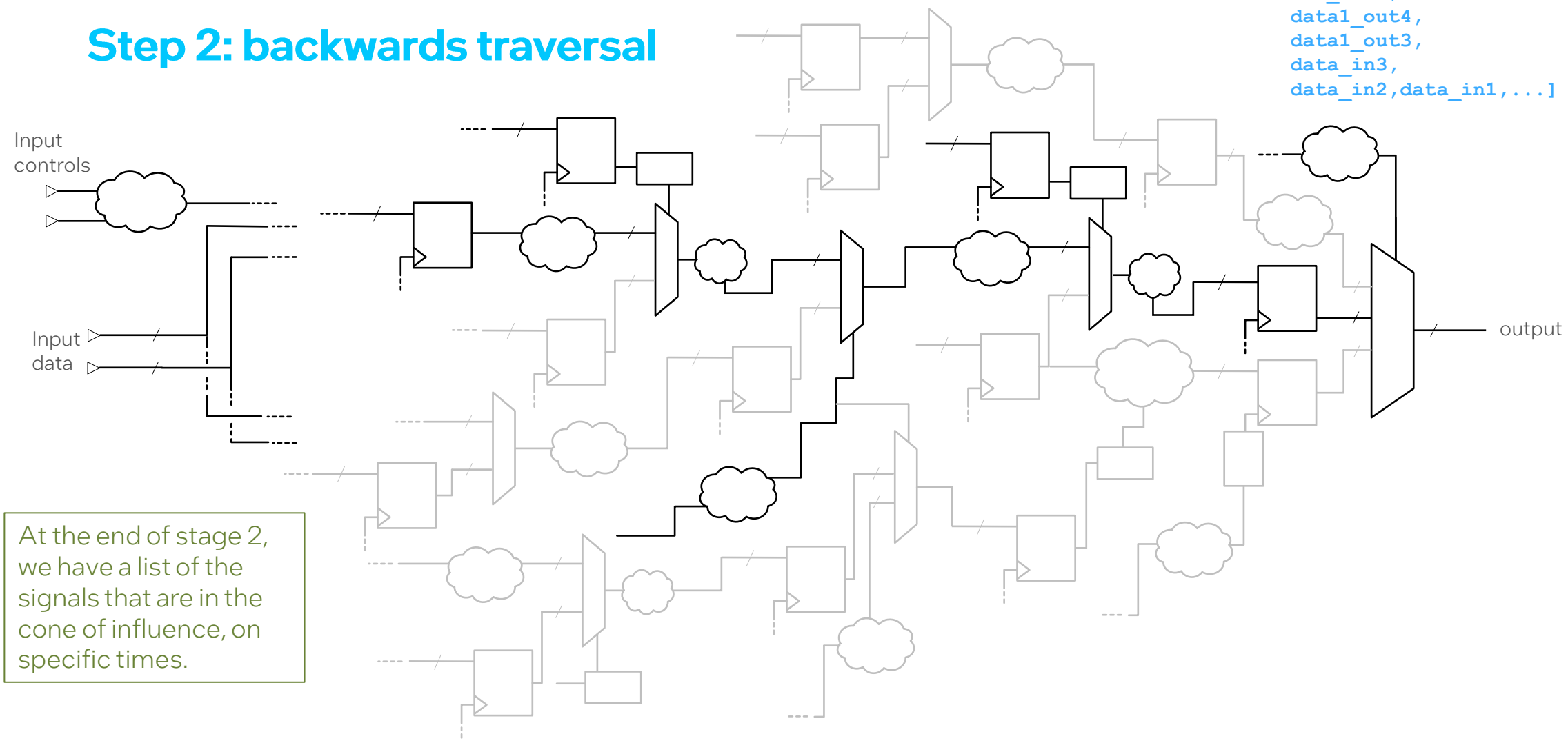




Timed Causal Fanin Analysis

Step 2: backwards traversal

```
Causal fanin:
[output,
ctl_out4,
data1_out4,
data1_out3,
data_in3,
data_in2,data_in1,...]
```

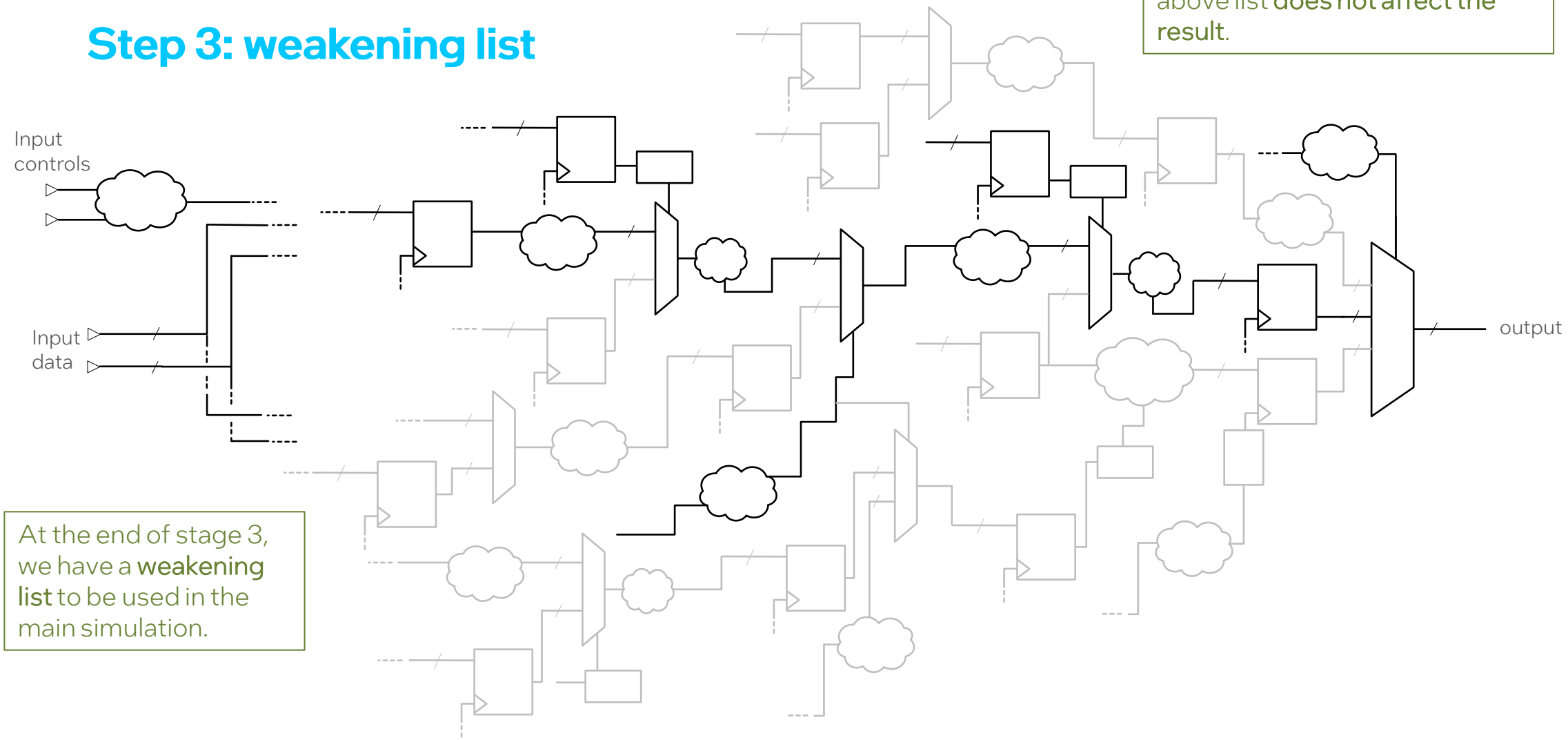


At the end of stage 2, we have a list of the signals that are in the cone of influence, on specific times.

Timed Causal Fanin Analysis

Step 3: weakening list

We can safely assume that everything that is NOT on the above list **does not affect the result**.



At the end of stage 3, we have a **weakening list** to be used in the main simulation.

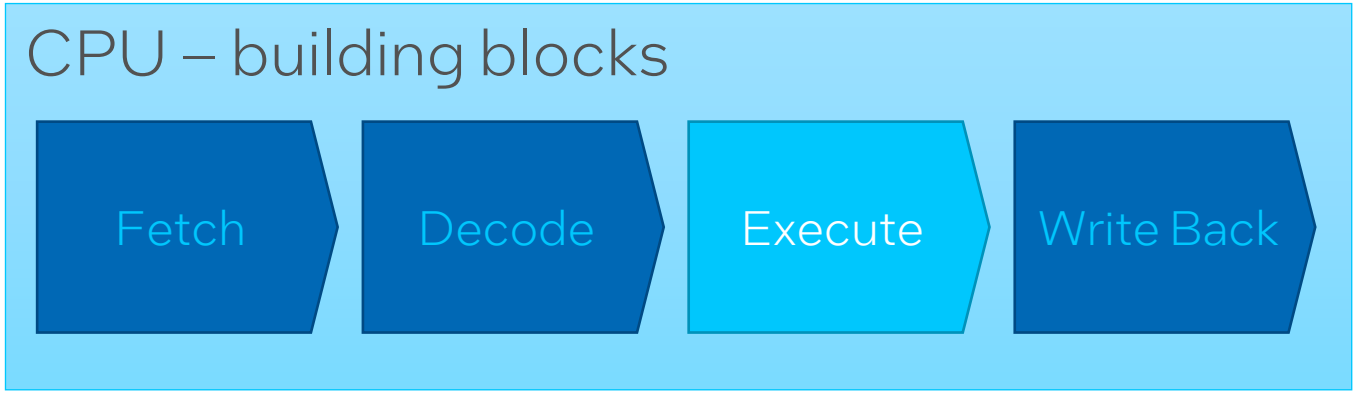
Debug process

The human user experience

- Computing causal fanin cone
  - Visible to the user
  - Understanding why a signal is there
- Tuning threshold for stages 1 and 2
  - Dynamic Weakening Limit
    - Too low --> fewer constant values for causal fanin traversal analysis
    - Too high --> heavier initial simulation
  - Causal Fanin Max-size / Min-Tick
  - Non-Causal Fanout Max-size / Min-Tick

FP16 Verification

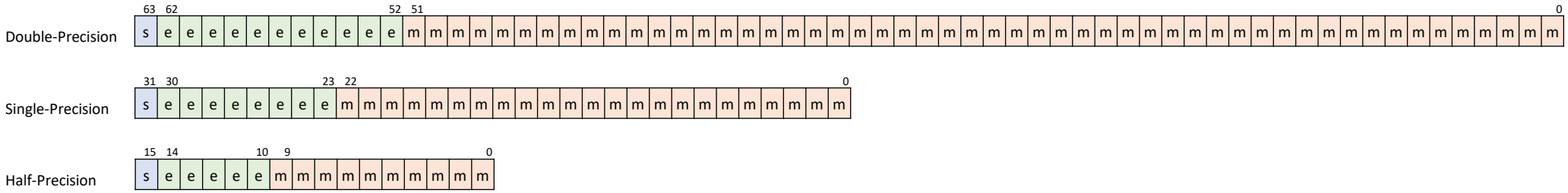
FP16 Verification



- All operations are fully verified using symbolic simulation.
- Complex arithmetic data-paths are especially risky: bugs are potentially customer visible and un-patchable

FP16 Verification

- Floating point numbers: $f = (-1)^s \times (1.m) \times 2^{(e-bias)}$
- Traditional floating-point types: single, double and extended precision floats.

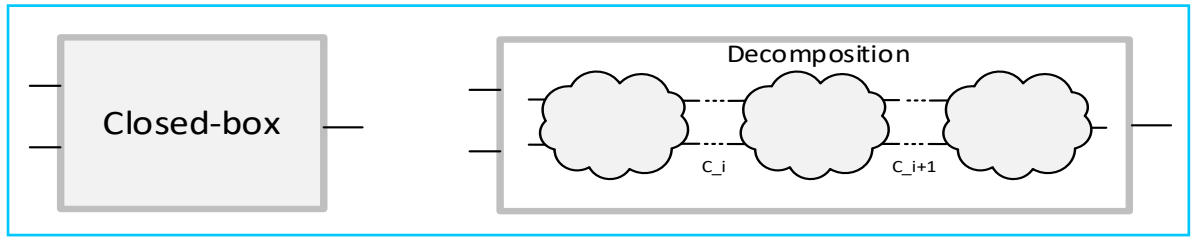


A new half-precision floating point type was introduced in recent projects.

* Mantissa size affects most on complexity

Different complexity challenges: “simple”, FMUL, FMA, FDIV operations

FP16 Verification – Closed-Box



- Specification is well-defined
- No need of insight into design implementation details
- Low sensitivity to internal design changes

Timed causal fanin analysis was a key enabler to the verification complexity reduction of FP16 operations. For the first time, we managed to prove **all complex operations closed-box**!

FP16 Verification and Timed Causal Fanin Analysis

- "Simple" uops (FCOM, FADD, conversions…)
  - Separate simple datapaths from complex ones
- FMUL
  - Isolate FMUL from FMA on the shared datapath (esp. rounding)
- FMA
  - Case split needed for complexity reduction
  - Remove logic that is not relevant to a specific case
- FDIV and FSQRT
  - Long-latency iterative operations, struggle against expression growth
  - Separate FP16 operation from higher precision datapaths

Summary

Timed causal fanin analysis

- Used extensively over many years
- Can do vs. cannot do (not fast vs. slow)
- Combines COI and constant-based reduction on timed basis
- Provides visibility to the human user
- Key complexity reduction technique in all Intel's symbolic simulation based FV work


intel

Q&A

Backup

- Parametric substitution
- Debug process

Parametric Substitution

How does it work?

$C(\vec{v})$: a conjunction of all verification assumptions.
Every symbolic variable in $\vec{v}$ is replaced with either a constant or a symbolic expression, such that $C(\vec{p})$ is satisfied.
… this allows us to simulate **exactly the set of values for which C is true**, in a single simulation.

Examples:

$$C_1(a, b) = a \wedge b$$

$$\vec{p} = [a \to 1, b \to 1]$$

$$C_1(\vec{p}) = true$$

$$C_2(a, b) = a \vee b$$

$$\vec{p} = \left[a \to a', b \to \left(b' \vee (\sim a')\right)\right]$$

$$C_2(\vec{p}) = a' \vee \left(b' \vee (\sim a')\right) = true$$

a′	b′	a'	b′|!a′
0	0	0	1
0	1	0	1
1	0	1	0
1	1	1	1

Parametric Substitution

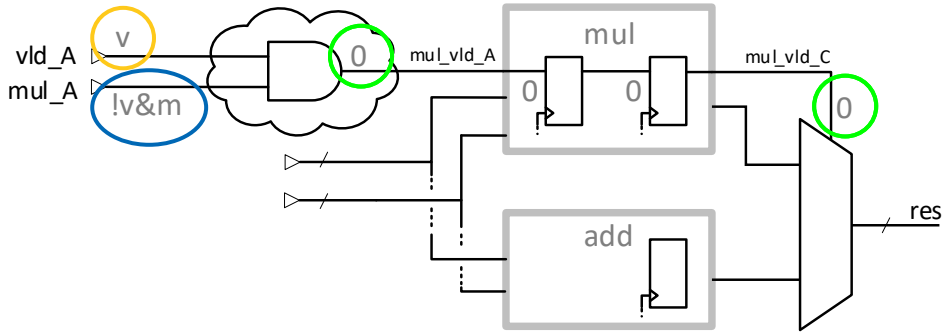
Single uop verification – Scope reduction by triggers

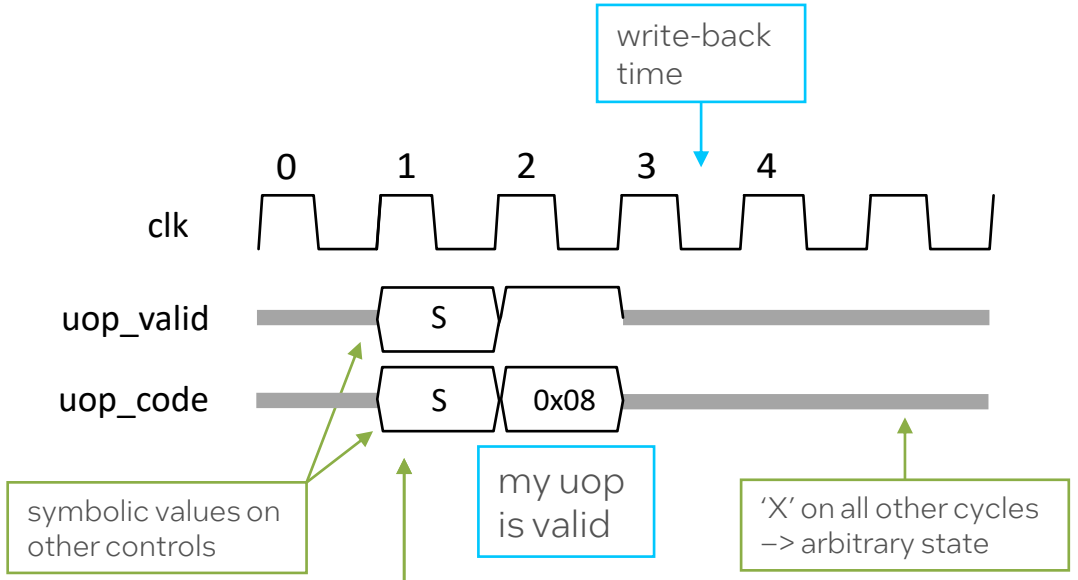
We cannot have two uops writing back at the same time
==
We cannot have:
a valid MUL 2-cyc uop at cycle 1
AND
a valid ADD 1-cyc uop at cycle 2

Parametric substitution allows this condition to transform to constant values in the simulation:

$$C_1(v,m) = (v \wedge \neg m) \qquad C_2(v,m) = \neg(v \wedge m)$$

$$C = C_1 \wedge C_2 \qquad \vec{p} = \left[v \to v', m \to \left((\sim v') \wedge m'\right)\right]$$





we can have other uops, as long as they satisfy the scheduling restrictions

Debug process

The human user experience

- Computing causal fanin cone
  - Visible to the user
  - Understanding why a signal is there

```
rSTE_debug_ncfow_show_statistics show_detail->T;
  ...
  Tick 6 fanin signal hierarchies:
      add/                118
      add/w2gadd_17_10/   30
      /                   10
  ...
  Tick 3 fanin signal hierarchies:
      /      114
      add/   9
      mul/   271
```

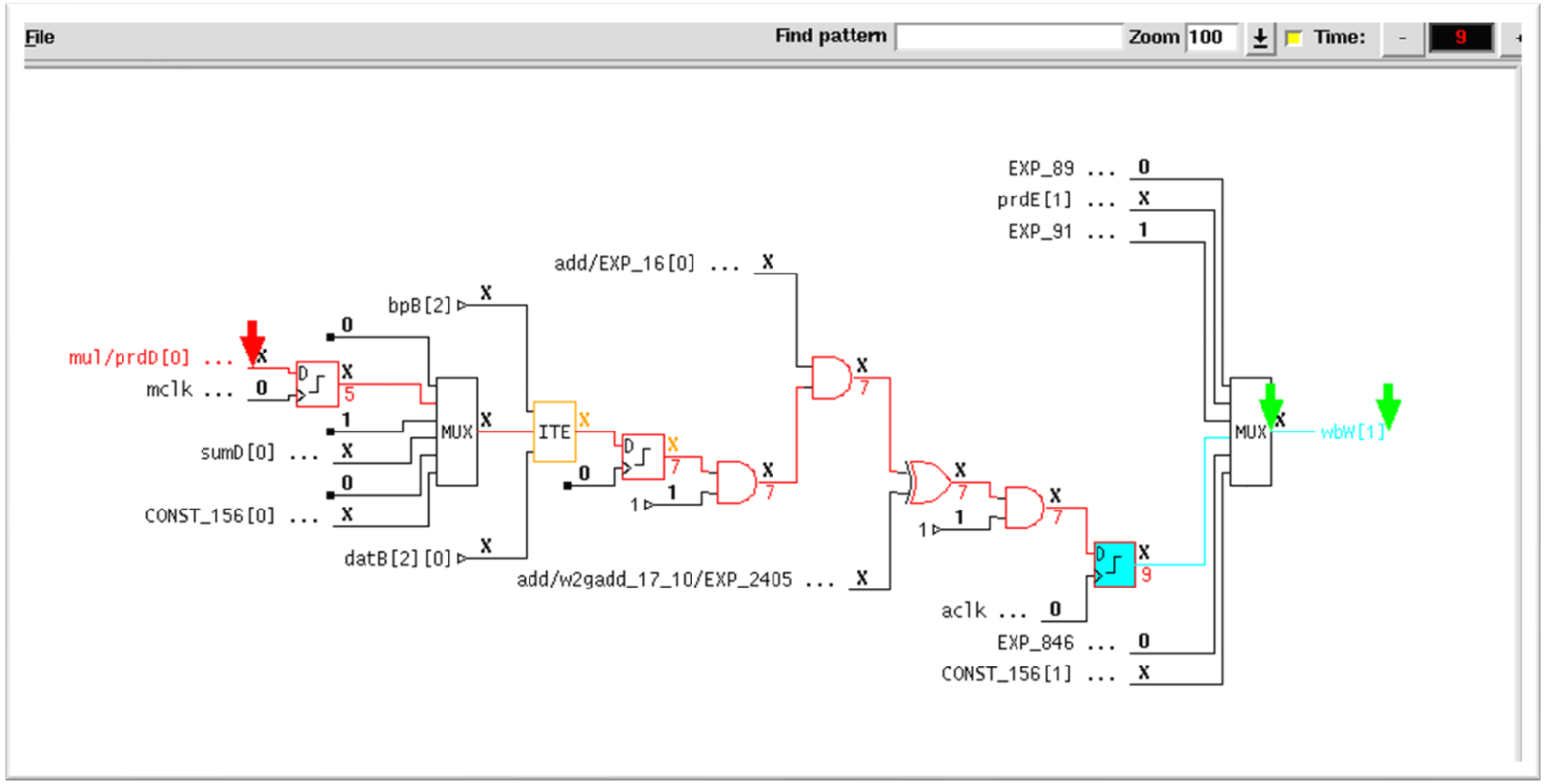
```
rSTE_debug_ncfow_get_causal_fanin ticks->[3] signal_prefix->"mul/";

[ ("mul/prdD[0]", 3), ("mul/prdD[1]", 3), ... ]
```

```
rSTE_debug_ncfow_get_causal_path ("mul/prdD[0]", 3);

[ ("wbW[1]", 9), ("sumD[1]", 9), ("add/sumC[1]", 7), ("add/sumcC[1]", 7),
  ("add/w2gadd_17_10/EXP_2391", 7), ("add/EXP_19[0]", 7), ("add/dinC[2][0]", 7),
  ("dinB[2][0]", 5), ("wbW[0]", 5), ("prdE[0]", 5), ("mul/prdD[0]", 3)]
```

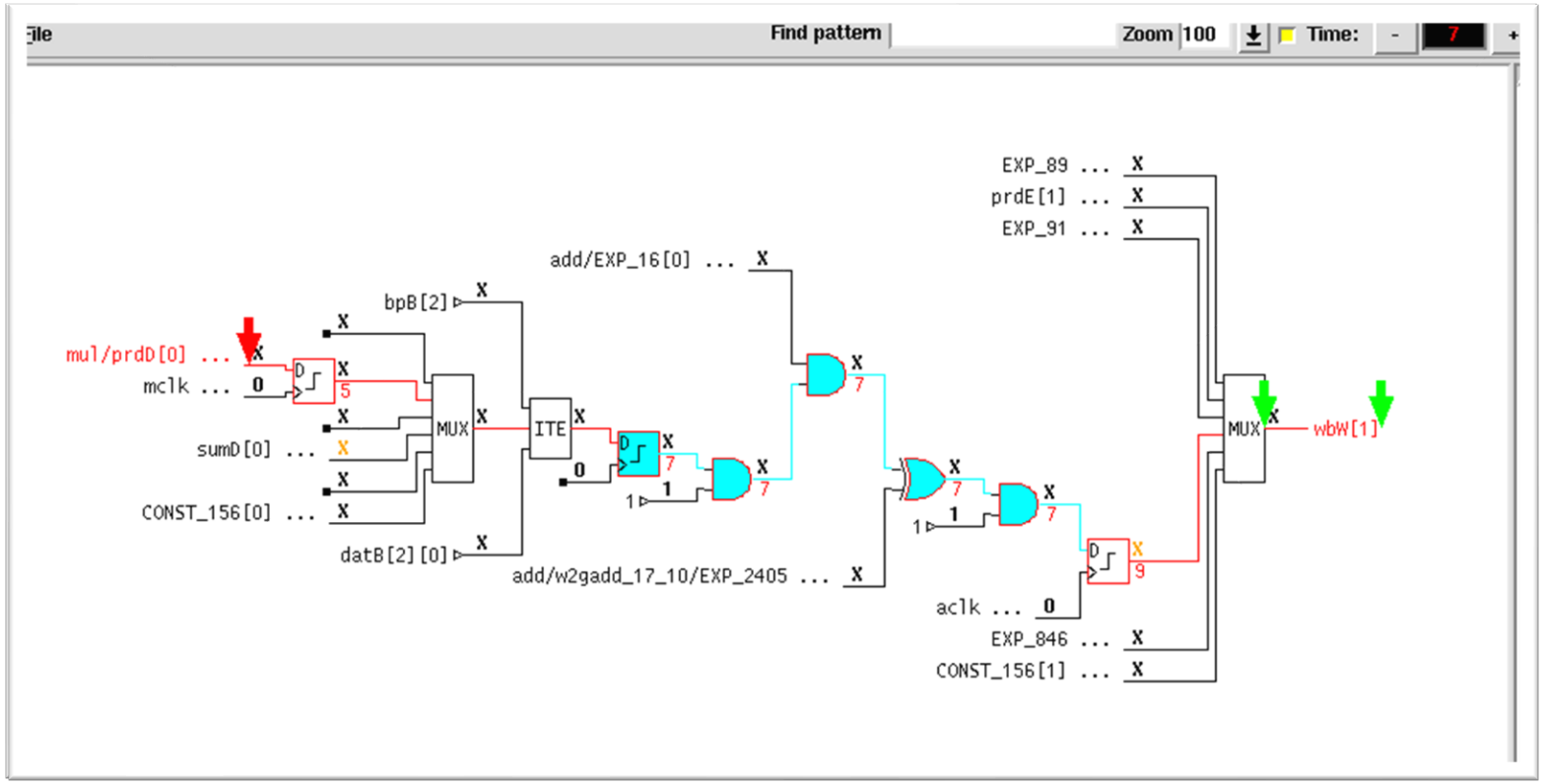
Debug process




```
rSTE_debug_ncfow_get_causal_path ("mul/prdD[0]", 3);

[ ("wbW[1]", 9), ("sumD[1]", 9), ("add/sumC[1]", 7), ("add/sumcC[1]", 7),
  ("add/w2gadd_17_10/EXP_2391", 7), ("add/EXP_19[0]", 7), ("add/dinC[2][0]", 7),
  ("dinB[2][0]", 5), ("wbW[0]", 5), ("prdE[0]", 5), ("mul/prdD[0]", 3)]
```

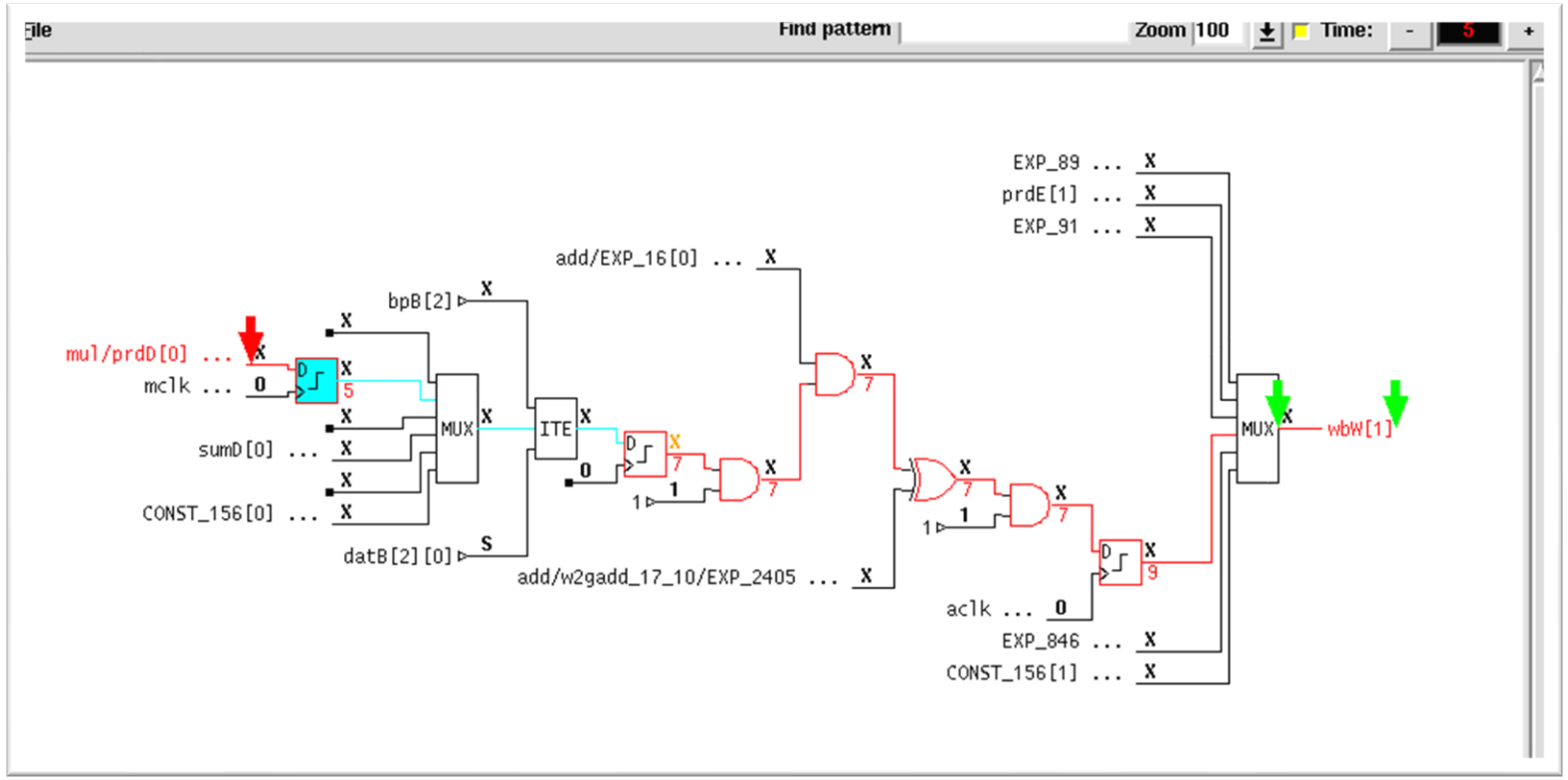
Debug process




```
rSTE_debug_ncfow_get_causal_path ("mul/prdD[0]", 3);

[ ("wbW[1]", 9), ("sumD[1]", 9), ("add/sumC[1]", 7), ("add/sumcC[1]", 7),
  ("add/w2gadd_17_10/EXP_2391", 7), ("add/EXP_19[0]", 7), ("add/dinC[2][0]", 7),
  ("dinB[2][0]", 5), ("wbW[0]", 5), ("prdE[0]", 5), ("mul/prdD[0]", 3)]
```

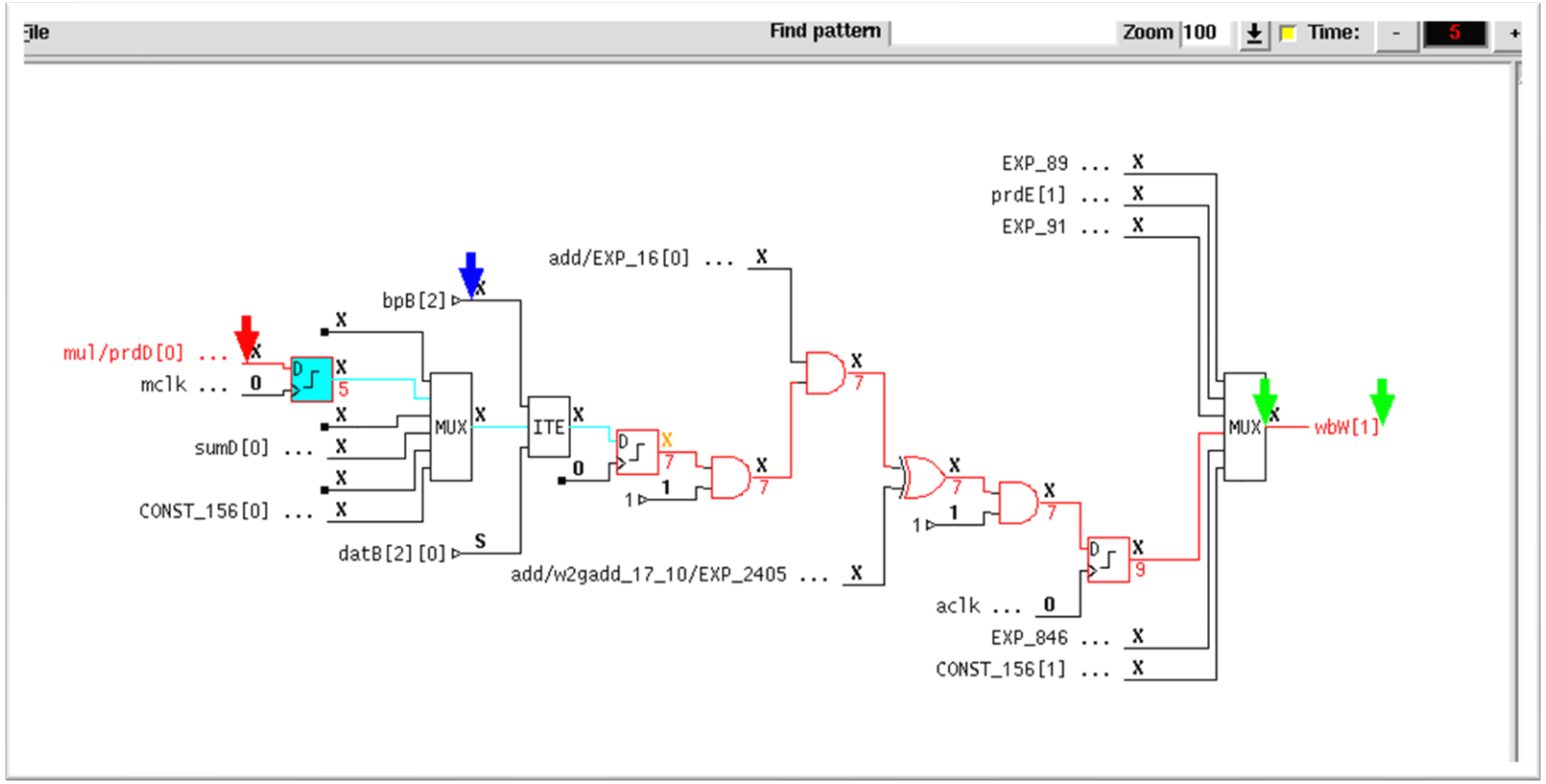
Debug process




```
rSTE_debug_ncfow_get_causal_path ("mul/prdD[0]", 3);

[ ("wbW[1]", 9), ("sumD[1]", 9), ("add/sumC[1]", 7), ("add/sumcC[1]", 7),
  ("add/w2gadd_17_10/EXP_2391", 7), ("add/EXP_19[0]", 7), ("add/dinC[2][0]", 7),
  ("dinB[2][0]", 5), ("wbW[0]", 5), ("prdE[0]", 5), ("mul/prdD[0]", 3)]
```

Debug process




```
rSTE_debug_ncfow_get_causal_path ("mul/prdD[0]", 3);

[ ("wbW[1]", 9), ("sumD[1]", 9), ("add/sumC[1]", 7), ("add/sumcC[1]", 7),
  ("add/w2gadd_17_10/EXP_2391", 7), ("add/EXP_19[0]", 7), ("add/dinC[2][0]", 7),
  ("dinB[2][0]", 5), ("wbW[0]", 5), ("prdE[0]", 5), ("mul/prdD[0]", 3)]
```