On-chip Dynamic Signal Sequence Slicing for Efficient Post-Silicon Debugging

Yeonbok Lee[1], Takeshi Matsumoto[2] , Masahiro Fujita[2,3]

[1] Department of Electronic Engineering, University of Tokyo

[2] VLSI Design and Education Center, University of Tokyo

[3] CREST, Japan Science and Technology Agency

Outline

- Introduction
- Proposed Method
- Experimental Results
- Summary and Future Works

Post-Silicon debugging

- **Post-Si Debugging?**
  - To find the root cause of an error detected by running fabricated chips
- **Post-Si debugging is getting mandatory**
  - Pre-silicon verification
    - Cannot be performed exhaustively
  - 71% of chip re-spins are due to **design bugs** [1]
    - → **Target of this work**
- **Post-Si debugging is even more time-consuming & critical**

[1] Miron Abramovici, et al.," A Reconfigurable Design-for-Debug Infrastructure for SoCs," DAC 2006

Problems in post-Si debugging

- **Execution traces are very LONG**
  - Running speed of a chip is very fast (x 10^6~10^9 than simulation)
  - Amount of data to be analyzed is very large
- **Limited observability / Controllability**
  - Requires scan chain or on-chip trace buffer

Scan-chain	• A snap-shot of the register values at each cycle can be obtained by one scanning-out • Available approach: 1. Get internal register values at a timing by scanning out 2. Re-simulate using the obtained values • Scanning-out once is time-consuming - "When to scan-out?" is an important problem
On-chip trace memory	• Real-time signal values for multiple cycles can be obtained • Limited size for area constraint (8K~256K)

Objective and contribution

- **Objective**
  - Propose a method to enhance the utility of scan-chain or on-chip trace memory
    - **Dynamic Signal Sequence Slicing Method**

      Extracts signal values relevant to target error from the original *LONG* signal sequence

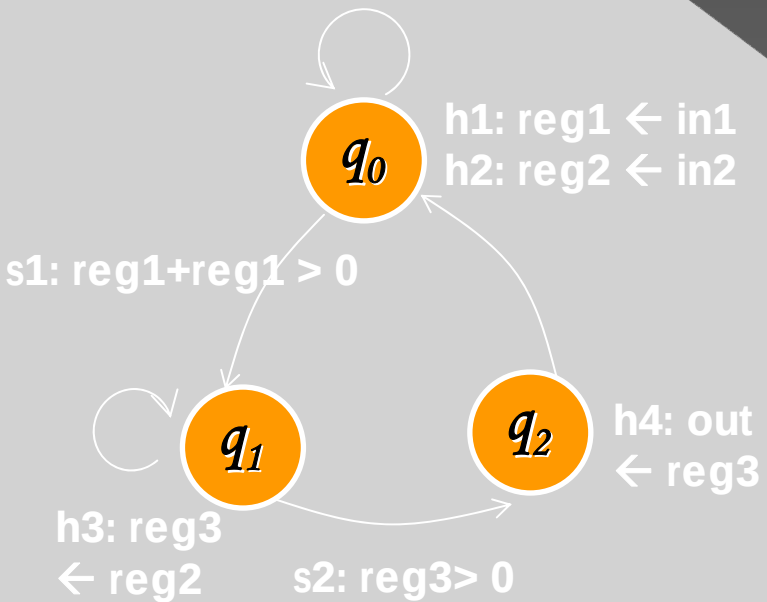
- **Contributions**
  - Save effort for analyzing irrelevant signal values
  - Enhance the efficiency of post-silicon debugging strategy based on simulation using scanned-out values
    - Supporting the decision of when to scan-out

Outline

- Introduction
- **Proposed Method**
- Experimental Results
- Summary and Future Works

Basic assumptions

- **Target design is described in FSMD(Finite State Machine with Data-path)**
  - i.e., Controller part and data-path part are identical
  - FSMD = $<Q, q_0, I, V, O, F, H>$



$Q = \{q_0, q_1, q_2\}$ **Control states**

$I = \{$ in1, in2 $\}$ **Input signals**

$V = \{$reg1, reg2, reg3$\}$ **Data registers**

$O = \{$out $\}$ **Output signals**

$F = \{ (q_0 \rightarrow q_0), (q_0 \times s1 \rightarrow q_1), (q_1 \rightarrow q_1), (q_1 \times s2 \rightarrow q_2), (q_2 \rightarrow q_0) \}$ **State transition functions**

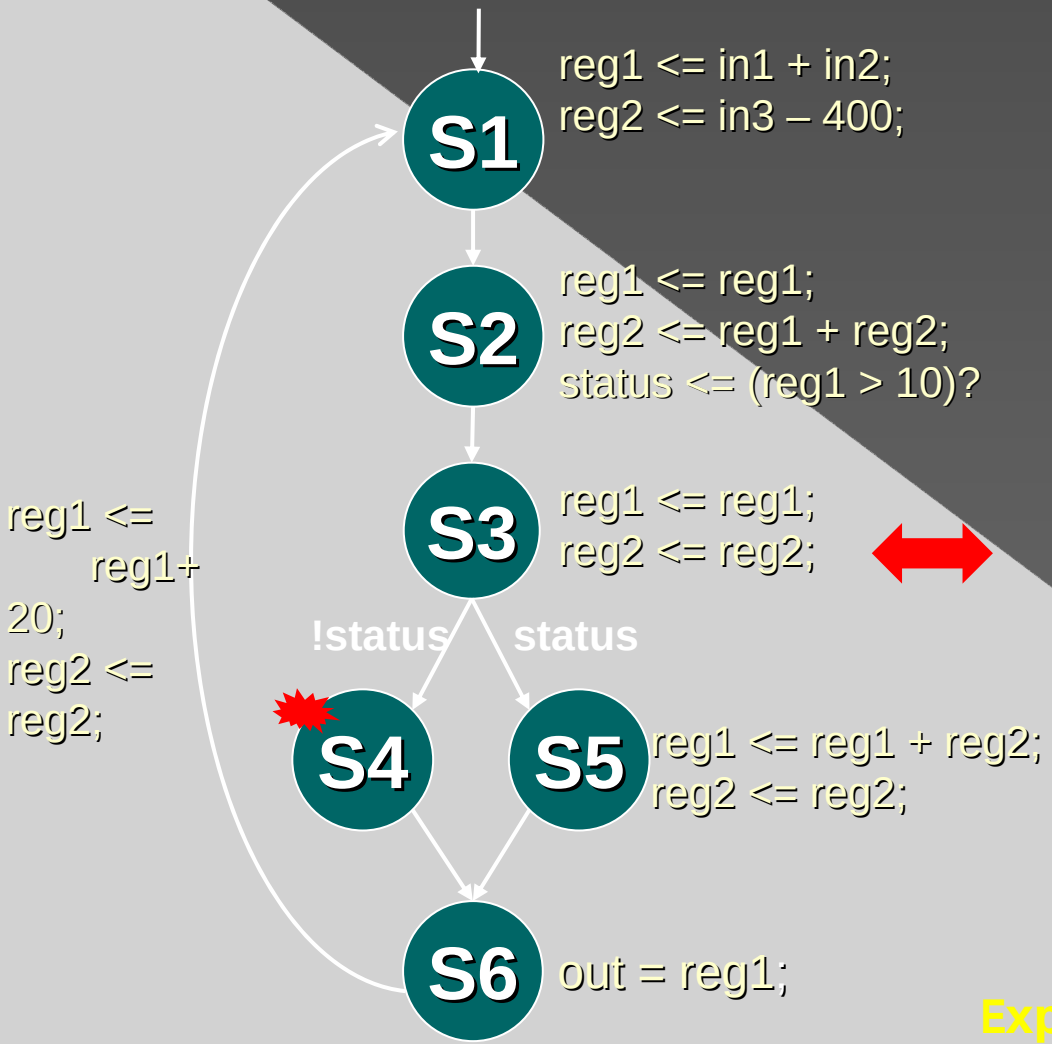
$H = \{ (q_0 \rightarrow (h1, h2)), (q_0 \times s1 \rightarrow (h3)), (q_1 \rightarrow (h3)), (q_1 \times s2 \rightarrow (h4)), (q_2 \rightarrow (h1, h2)) \}$ **Value assignment functions**

- **An erroneous (output) value and its timing are known**
  - E.g., by using assertion checker

Terminologies

- **Basic representation**
  - [*signal_name*](*t*) : the alue of the signal named *signal_name* at a clock cycle *t*
- **Definition of terminologies**
  - **Observing Signal Set** S
    - A set of signals that we want to observe.
    - $S = \{si \mid si \in I \cup V \cup O\}$
    - (Specified manually)
  - **Dynamic signal sequence slice of** $x(tn)$
    - Signal values having dependency to $x(tn)$ from each signal value sequence of si, *i.e.*, $\{si(0), \ldots, si(tn)\}$
  - **Dynamic signal sequence slicing for** $x(tn)$
    - To obtain the dynamic signal sequence slice of $x(tn)$

Illustrative Example : when S = I



Cycle	Ctrl State	in1	in2	in3	out
1	S1	20	30	100	X
2	S2	15	15	150	X
3	S3	10	20	100	X
4	S5	5	15	50	X
5	S6	0	10	0	-200
6	S1	1	5	2	X
7	S2	5	0	4	X
8	S3	10	5	6	X
9	S4	15	10	8	X
10	S6	20	15	10	26

Expected results to be obtained

FSMD(Finite State Machine)

An I/O Sequence

d-tag (Dependency Tag)

- **d-tag**
  - A new variable generate for each signal
    - A **d-tag** of signal s is represented as s_d
    - $s\_d(tn)$ represents whether a target signal value $o(tm)$ is dependent on $s(tn)$

Definition of d-tag

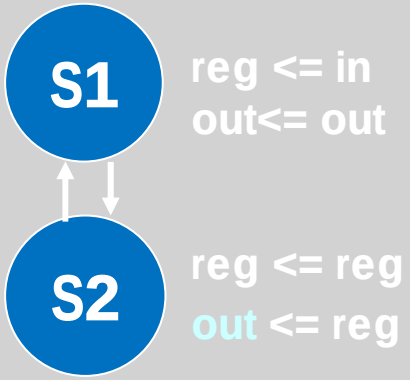
$$s\_d(tn) \begin{cases} 1 & \text{A signal value } o(tm) \text{ is dependent on } s(tn) \\ 0 & \text{Otherwise} \end{cases}$$

- Obviously, $o\_d(tm)$ for $o(tm)$ must be 1
- That is,
  - "To find the input signal values which the erroneous output value $o(tm)$ has dependency on"

    = "To find the input values when the d-tag value of them become 1 *w.r.t.* $o\_d(tm)$ specified as 1"

Computation of d-tags

- The value of $s\_d(t)$ of s :
  - Can be inferred by the d-tag values of the signals which use the $s(t)$ in the next timing $t+1$
- For example,



reg_d(t) = ((reg_d(t+1)=1) ∧(state(t+1) = S2))
∨ ((**out_d**(t+1)=1) ∧ (state(t+1)=S2))

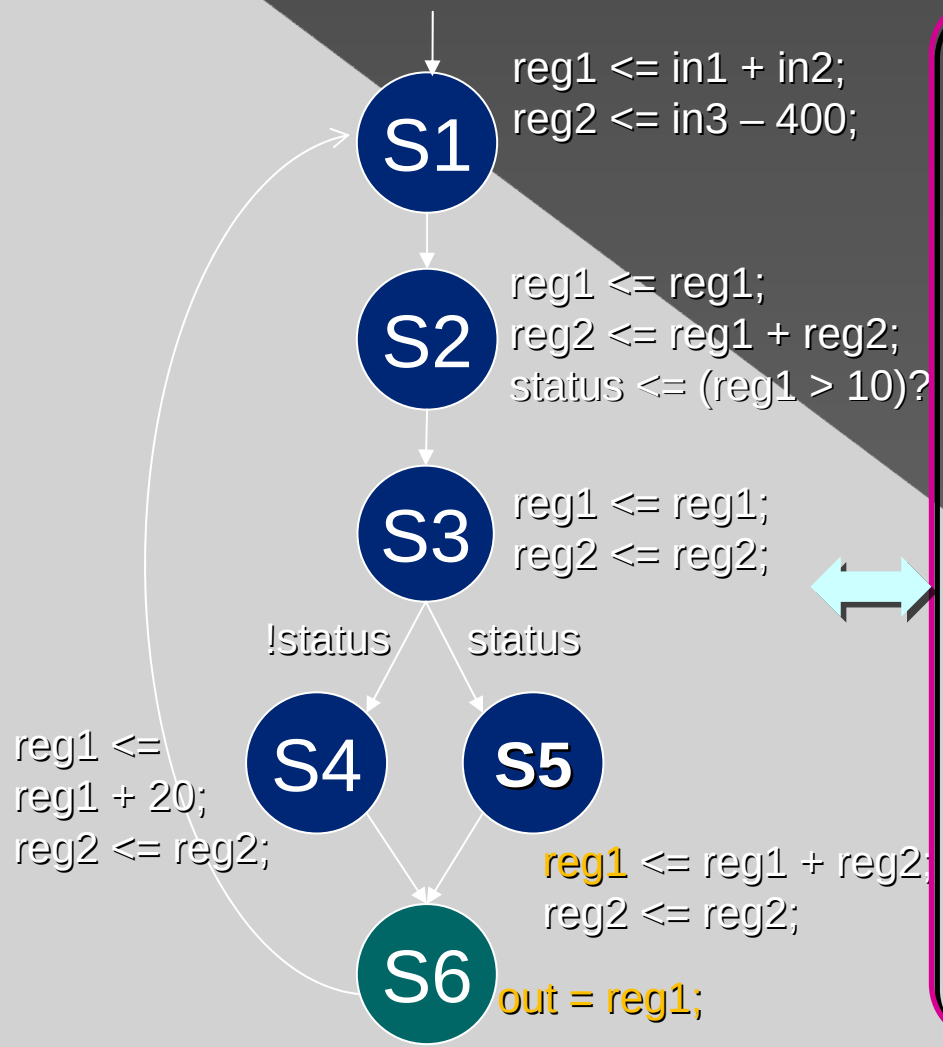
in_d(t) = (reg_d(t+1)=1) ∧ (state(t+1)=S1)

- Generalization
  - $s\_d(t) = \bigvee_{q} \left(\bigvee_{x\_d \mid x \leftarrow s \text{ at } q} x\_d(t+1) \right) \wedge \left(state(t+1) = q \right)$

Control state

  - Thus, other d-tag values can be obtained by computing the equations **backwardly** from a given $o\_d(tm)$

Example of d-tag computation



reg1_d(t) = (out_d (t+1) =1) ∧((state(t+1)=S6)
|| (reg1_d (t+1) =1) ∧
((state(t+1)=S5)
|| (reg1_d (t+1) =1) ∧
((state(t+1)=S4)
|| (reg1_d (t+1) =1) ∧
((state(t+1)=S3)

reg2_d (t) = (reg1_d (t+1) =1) ∧ ((state(t+1)=S5)
|| (reg1_d (t+1) =1) ∧
((state(t+1)=S1)
|| (reg2_d (t+1) =1)∧
((state(t+1)=S5)
|| (reg2_d (t+1) =1) ∧
((state(t+1)=S1)
|| (reg2_d (t+1) =1) ∧
((state(t+1)=S4)
|| (reg2_d (t+1) =1) ∧
((state(t+1)=S3)

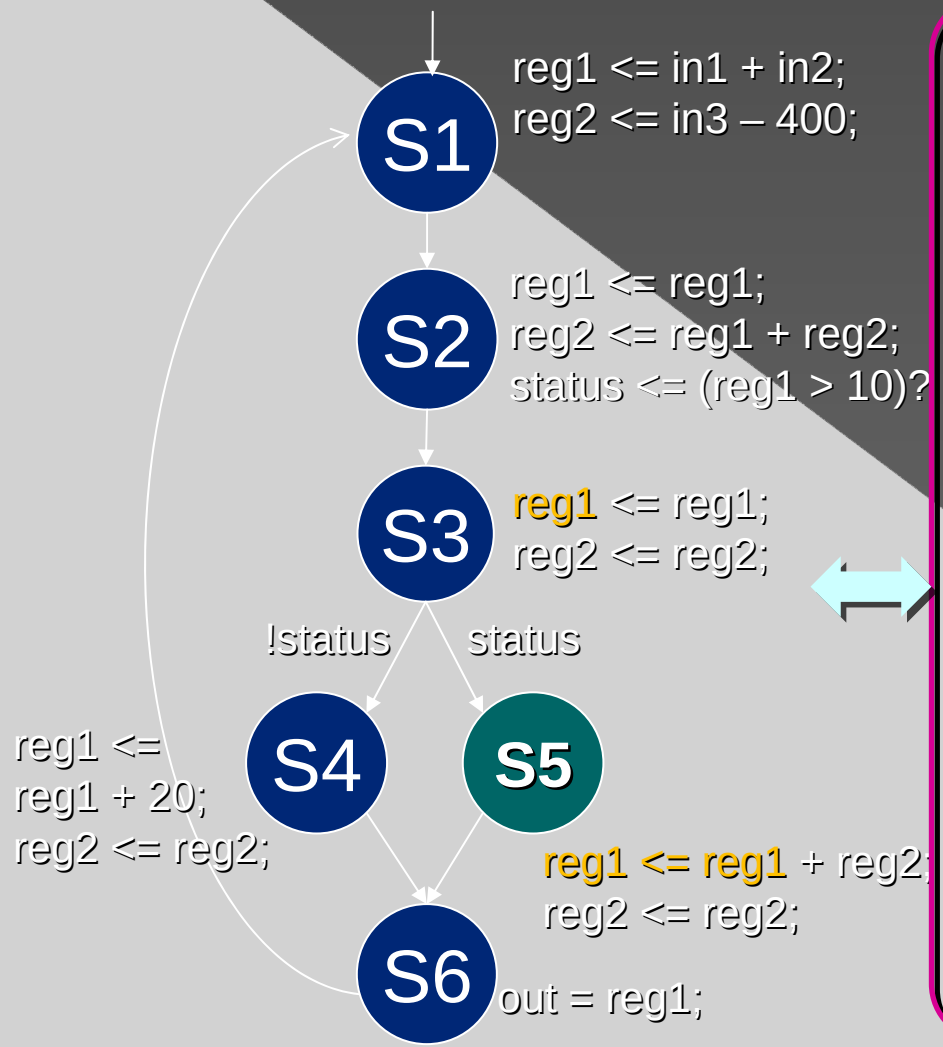
in1_d (t) = (reg1_d (t+1) =1) ∧
((state(t+1)=S1)
|| (reg2_d (t+1) =1) ∧
((state(t+1)=S1)

in2_d (t) = (reg1_d (t+1) =1) ∧
((state(t+1)=S1)

in3_d (t) = (reg2_d (t+1) =1) ∧
((state(t+1)=S1)

FSMD(Finite State Machine)

Example of d-tag computation



reg1_d(t) = (out_d (t+1) =1) ∧((state(t+1)=S6)
|| (reg1_d (t+1) =1) ∧
((state(t+1)=S5)
|| (reg1_d (t+1) =1) ∧
((state(t+1)=S4)
|| (reg1_d (t+1) =1) ∧
((state(t+1)=S3)

reg2_d (t) = (reg1_d (t+1) =1) ∧ ((state(t+1)=S5)
|| (reg1_d (t+1) =1) ∧
((state(t+1)=S1)
|| (reg2_d (t+1) =1)∧
((state(t+1)=S5)
|| (reg2_d (t+1) =1) ∧
((state(t+1)=S1)
|| (reg2_d (t+1) =1) ∧
((state(t+1)=S4)
|| (reg2_d (t+1) =1) ∧
((state(t+1)=S3)

in1_d (t) = (reg1_d (t+1) =1) ∧
((state(t+1)=S1)
|| (reg2_d (t+1) =1) ∧
((state(t+1)=S2)

in2_d (t) = (reg1_d (t+1) =1) ∧
((state(t+1)=S1)

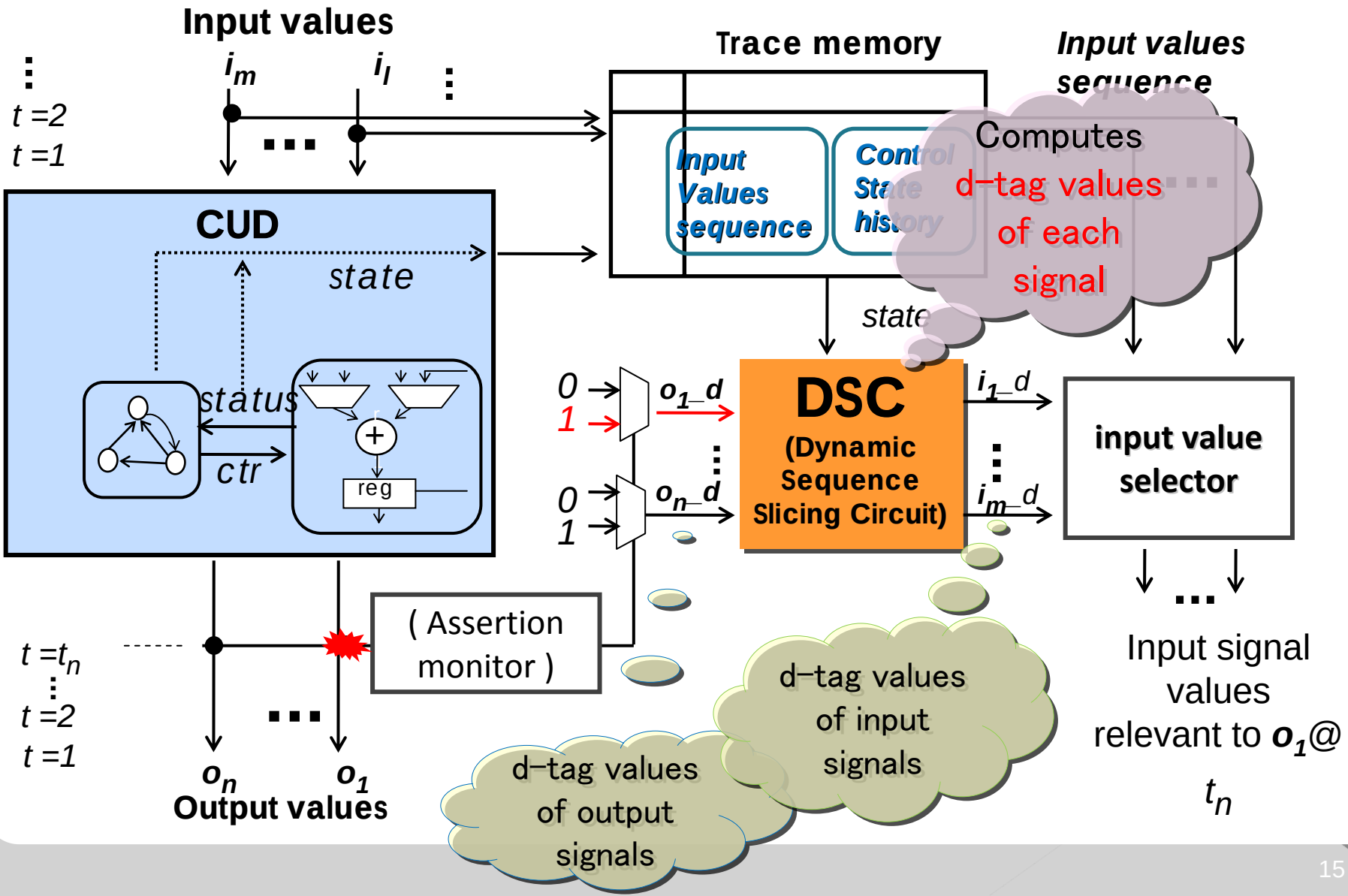
in3_d (t) = (reg2_d (t+1) =1) ∧
((state(t+1)=S1)

FSMD(Finite State Machine)

Dynamic signal sequence slicing

- Basic procedure
  1. Prepare equations for computing d-tags of signals in CUD (* CUD : Circuit Under Debugging)
  2. In the erroneous execution, *
     - Preserve the followings :
       - State transition history
       - Input signal value sequence
     - When the output signal becomes error:
       - Set its d-tag of to 1 (here, t = tm)
  3. Compute the d-tags backwardly from t = tm toward t=0
     - Return the input signals when the d-tags of them become 1 as result
- Can be realized either by SW or HW

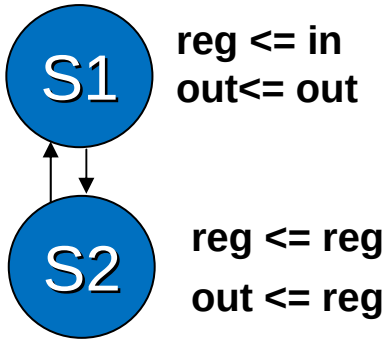
An on-chip Implementation of Dynamic Signal Sequence Slicing



Inputs and Outputs (when $S = I$) - for the on-chip implementation

- **Required items**
  - Data to be preserved in on-chip trace memory
    - Input signal value sequences
    - d-tag values of the output signals (1 when error)
    - Control state transition history
  - Additional Circuitry
    - For DSC circuit
      - FFs to preserve the d-tag values (0/1) of the signals in CUD
      - Combination circuits for computing the d-tags
    - A circuitry for extracting the input signal values when the d-tags of them become 1
- **The output results**
  - Input signal values that are relevant to the target error

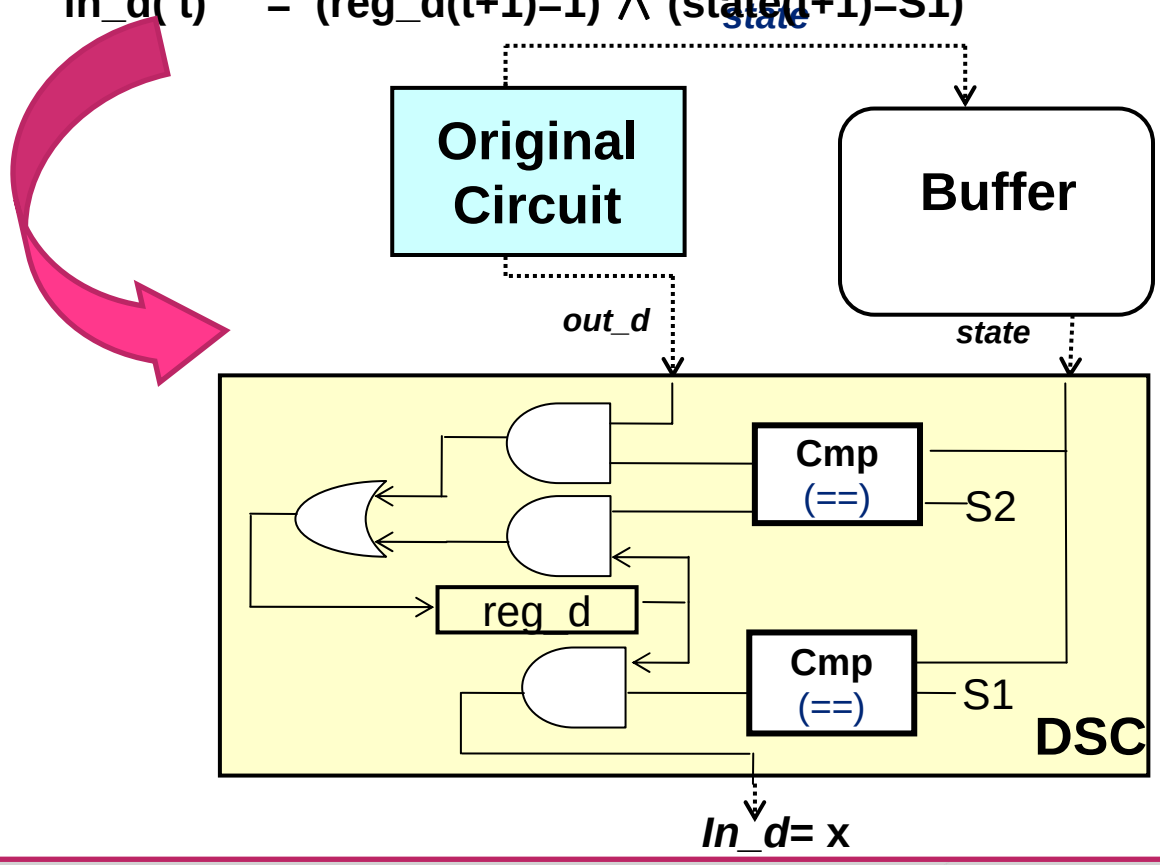
An example of DSC



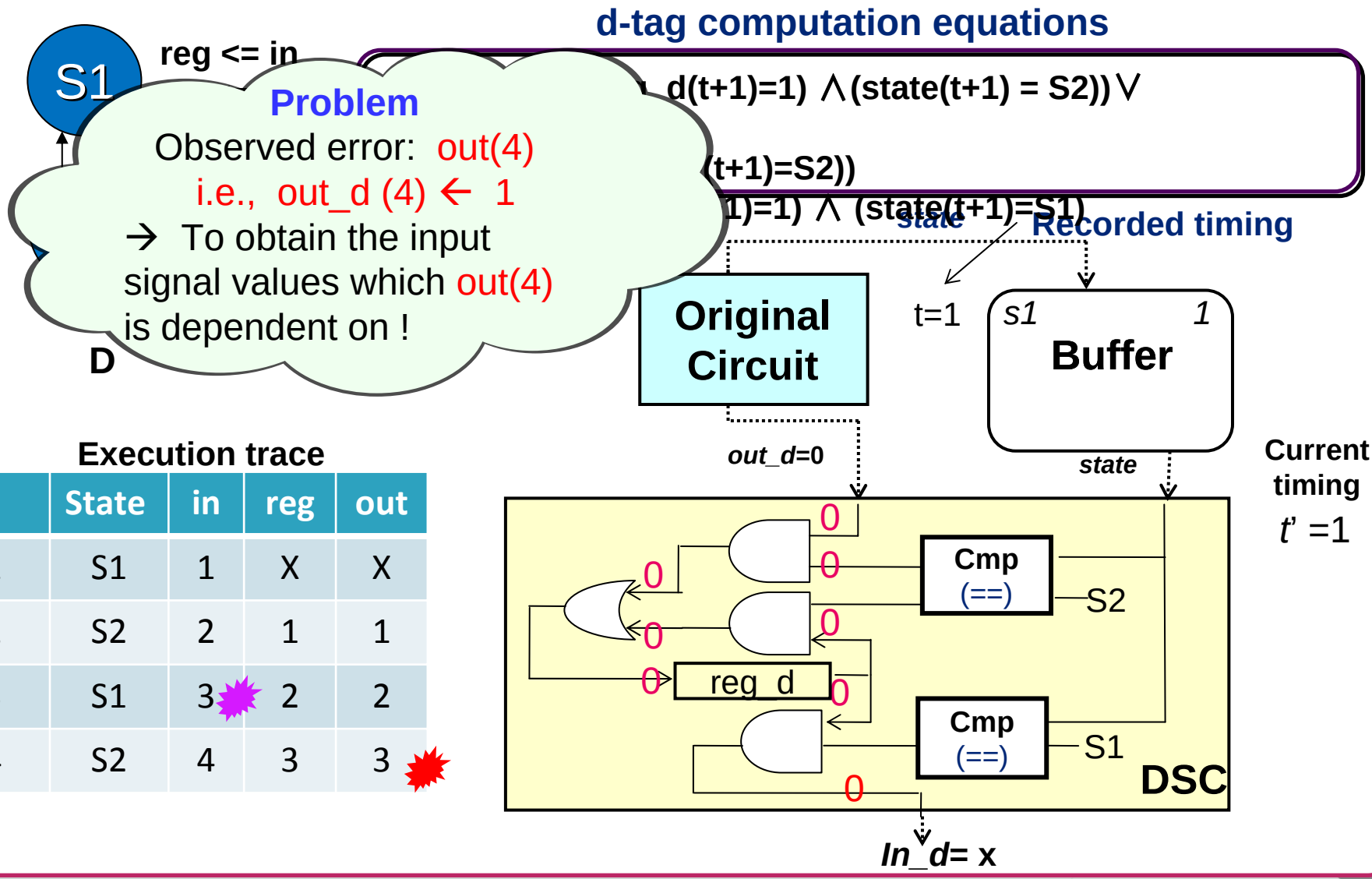
d-tag computation equations

$reg\_d(t) = ((reg\_d(t+1)=1) \wedge (state(t+1) = S2)) \vee ((out\_d(t+1)=1) \wedge (state(t+1)=S2))$

$in\_d(t) = (reg\_d(t+1)=1) \wedge (state(t+1)=S1)$



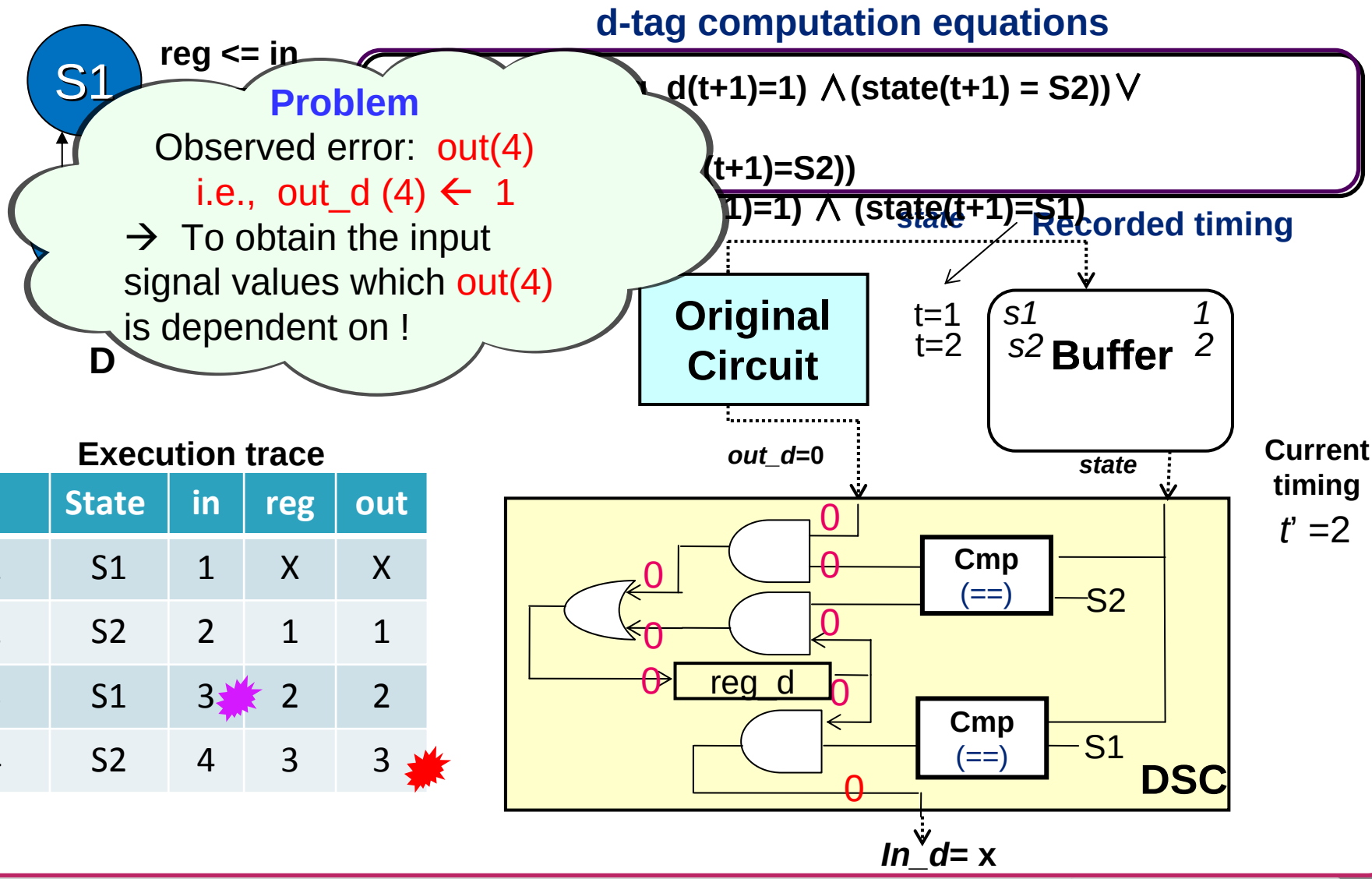
An execution example



Execution trace

t	State	in	reg	out
1	S1	1	X	X
2	S2	2	1	1
3	S1	3	2	2
4	S2	4	3	3

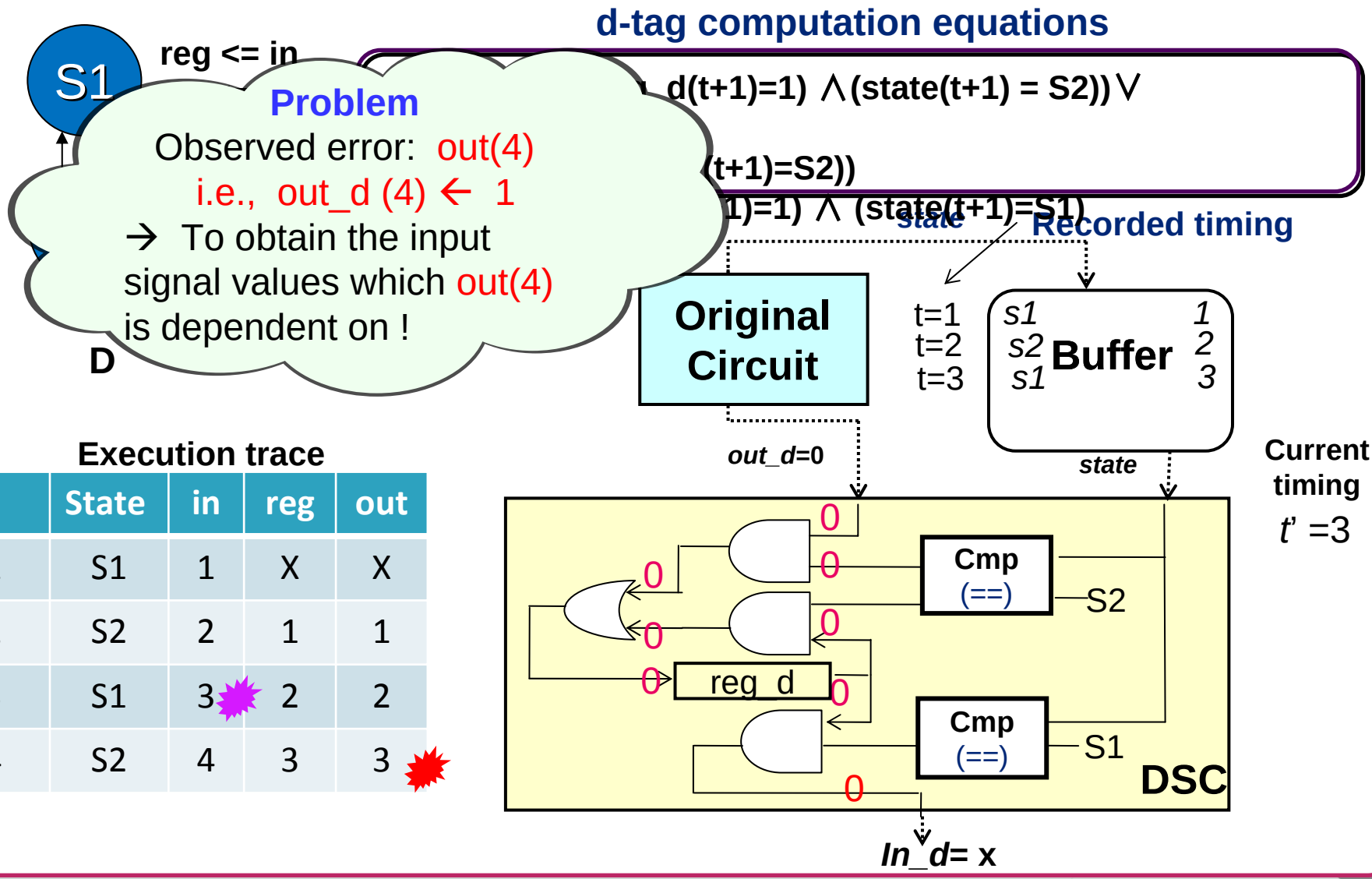
An execution example



Execution trace

t	State	in	reg	out
1	S1	1	X	X
2	S2	2	1	1
3	S1	3	2	2
4	S2	4	3	3

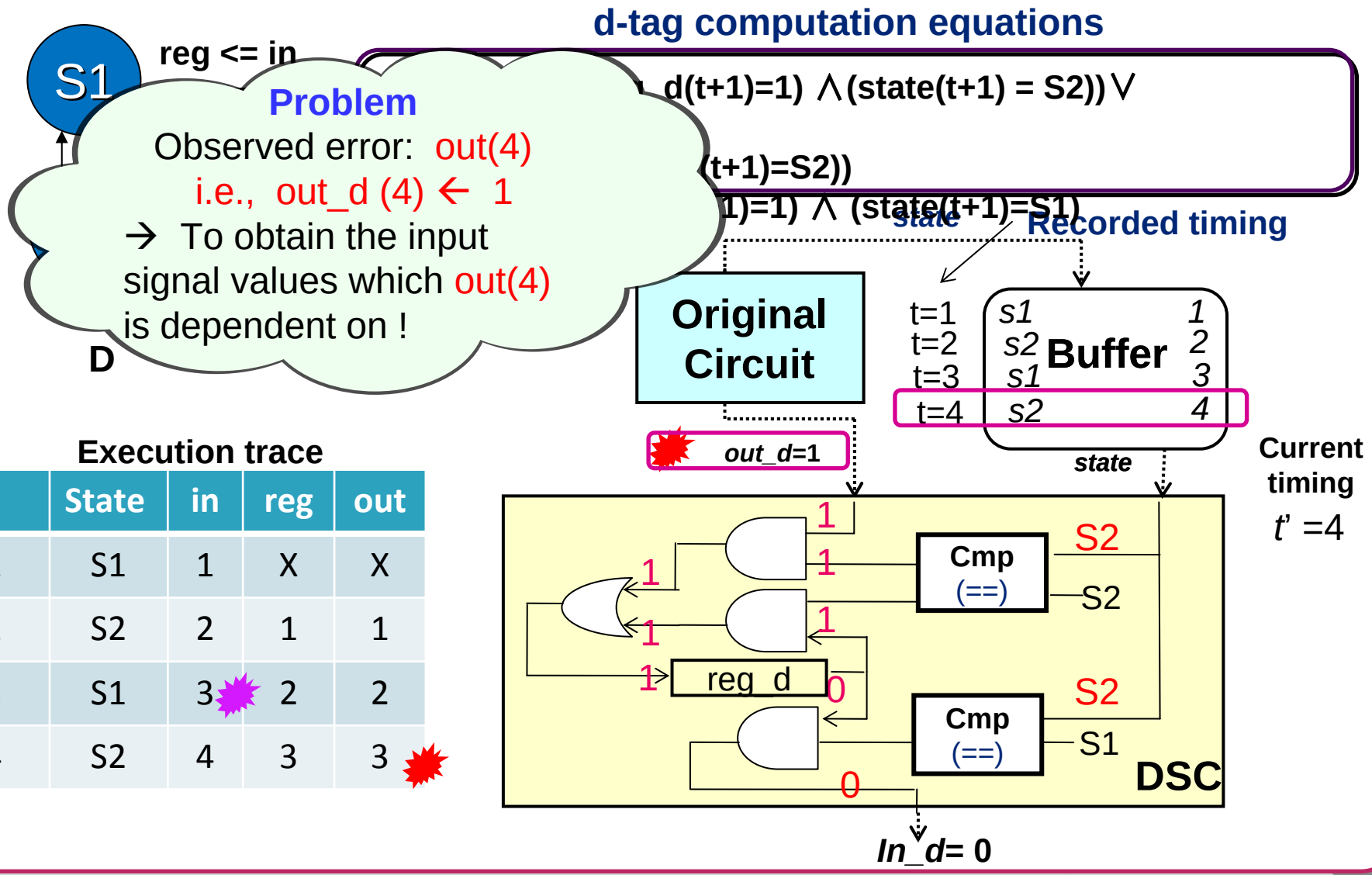
An execution example



Execution trace

t	State	in	reg	out
1	S1	1	X	X
2	S2	2	1	1
3	S1	3	2	2
4	S2	4	3	3

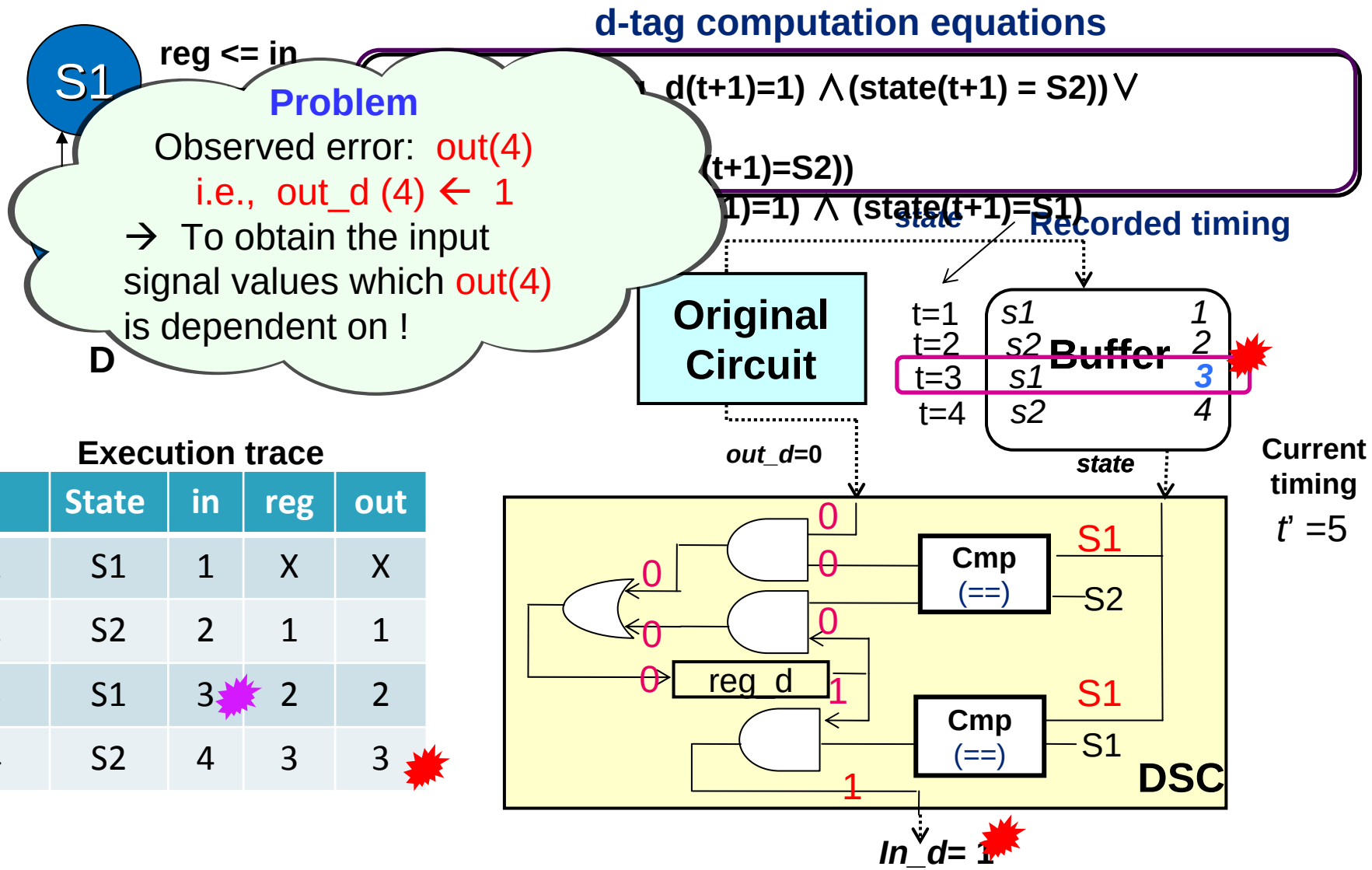
An execution example



Execution trace

t	State	in	reg	out
1	S1	1	X	X
2	S2	2	1	1
3	S1	3	2	2
4	S2	4	3	3

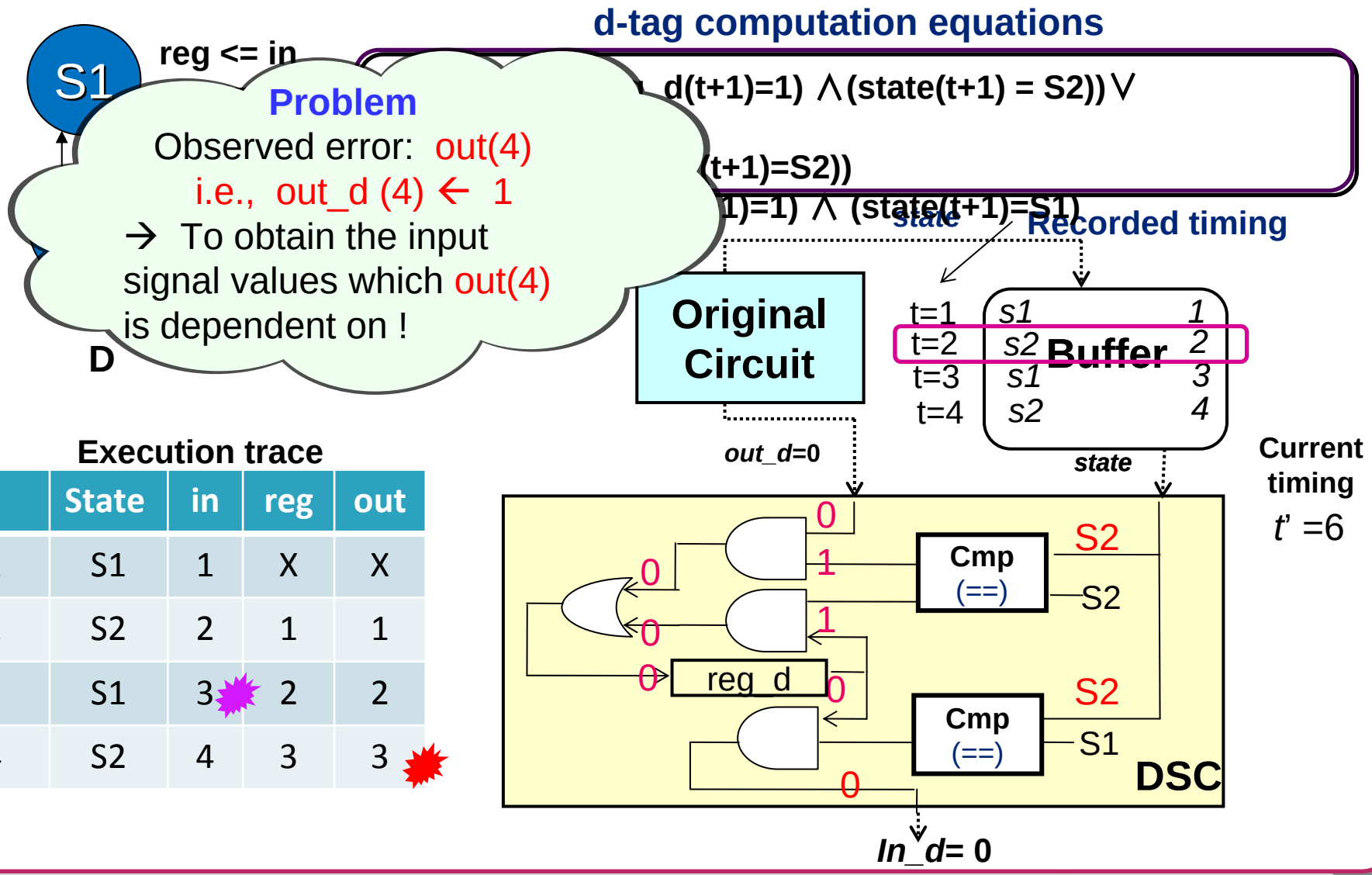
An execution example



Execution trace

t	State	in	reg	out
1	S1	1	X	X
2	S2	2	1	1
3	S1	3	2	2
4	S2	4	3	3

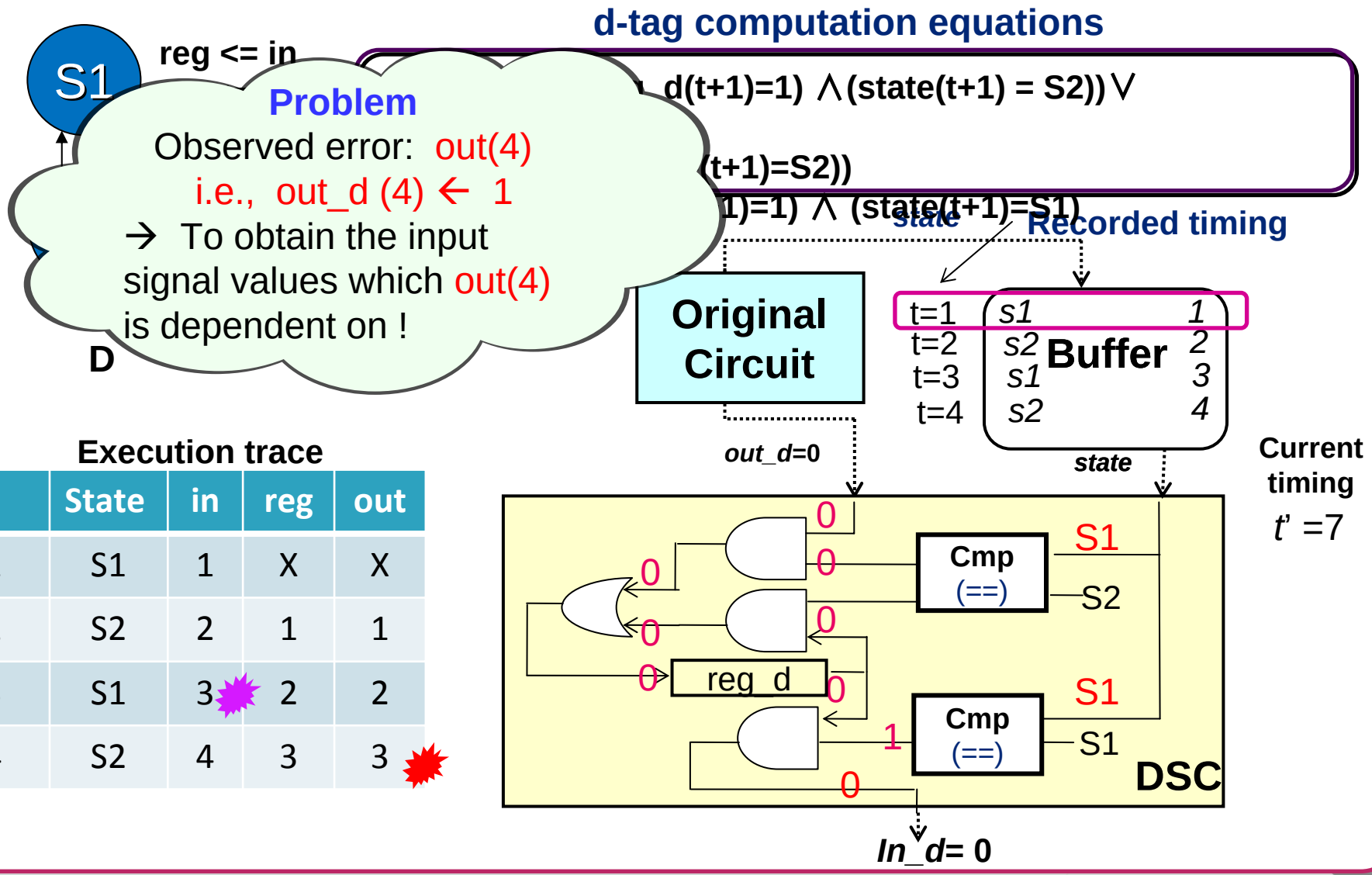
An execution example



Execution trace

t	State	in	reg	out
1	S1	1	X	X
2	S2	2	1	1
3	S1	3	2	2
4	S2	4	3	3

An execution example



Execution trace

t	State	in	reg	out
1	S1	1	X	X
2	S2	2	1	1
3	S1	3	2	2
4	S2	4	3	3

Outline

- Introduction
- Proposed Method
- **Experimental Results**
- Summary and Future Works

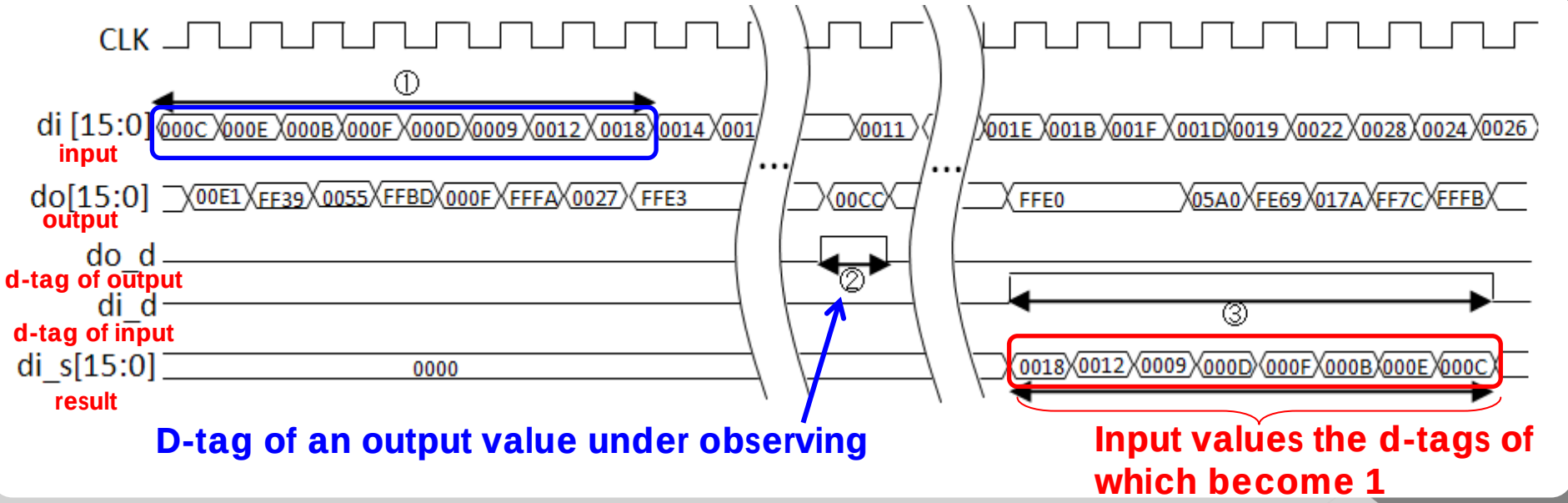
Experiment

- **Experiment**
  - Implemented on-chip dynamic signal sequence slicing method on example designs
  - Evaluated the functionality and area overhead
- **Example designs (RTL)**
  - Simple : A design that performs simple arithmetic computation(362 lines)
  - Ellip : An elliptical filter design (1182 lines)
  - IDCT : 8x8 IDCT (1722 lines)
- **Simulator**
  - Cadence Verilog-XL
- **RTL synthesizer**
  - Synopsys Design Compiler

Experimental Results

	# of input events			# of Gates		
	Original	Extracted	Reduction rate	Original	DSC	Overhead
Simple	30,000	5,000	83%	2758	156	5.65%
Ellip	10,000	769	93%	16423	773	4.71%
IDCT	10,000	64	99%<	23672	462	1.95%

Waveform of 8x8 IDCT design simulation and the slicing results



✓ Furthermore, we implemented the designs on an FPGA board

: Going to evaluate using “very long” input sequences

Outline

- Introduction
- Proposed Method
- Experimental Results
- **Summary and Future Works**

Summary and Future Works

- **Summary**
  - Proposed a new variable d-tag representing the dynamic dependencies among signals in CUD
  - Proposed dynamic signal sequence slicing method based on d-tag computation
  - Proposed an on-chip implementation of dynamic signal sequence slicing method
  - Evaluated functionality and area overhead using 3 designs
    - Relevant signal values to a specified signal value are extracted
    - Area overhead of the additional circuitry was 4 % in average
- **Future work**
  - To examine various implementations
  - To evaluate using “very long execution”