Snap-SAT: A One-Shot Energy-Performance-Aware All-Digital Compute-in-Memory Solver for Large-Scale Hard Boolean Satisfiability Problems

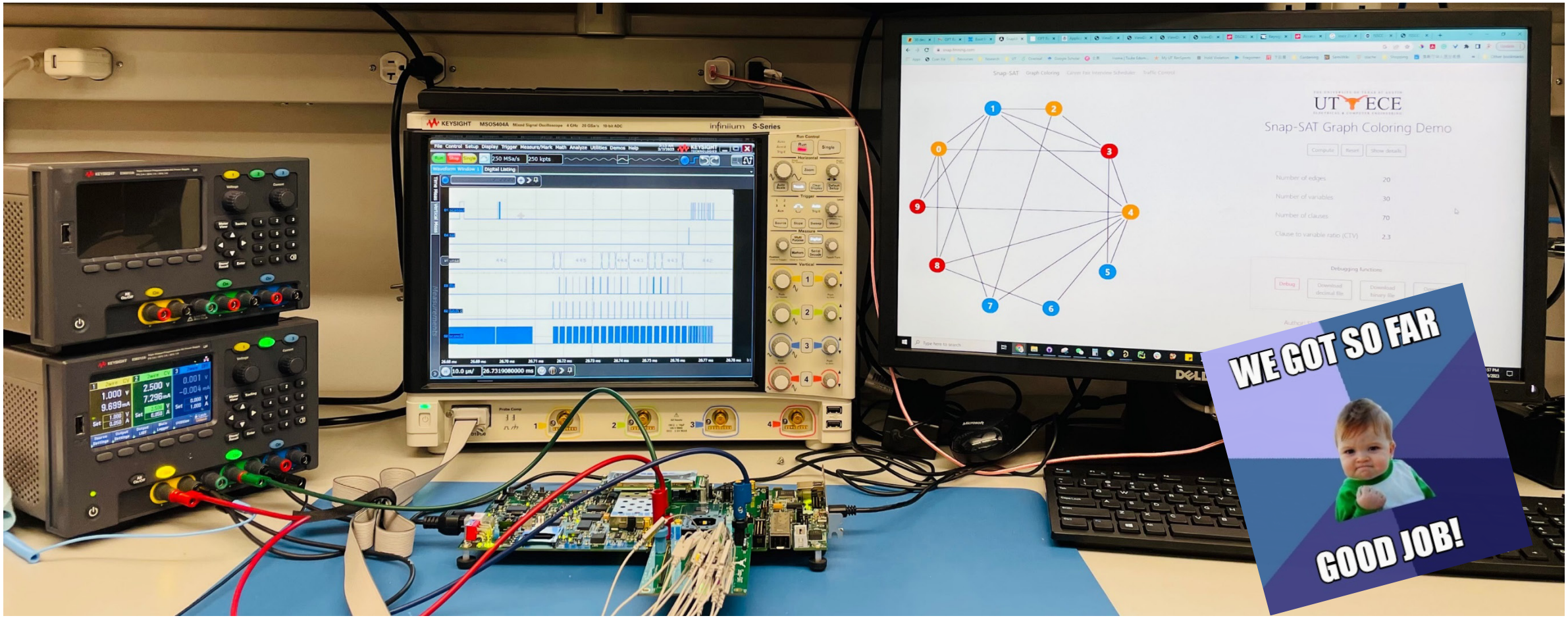
Shanshan Xie, Mengtian Yang, S. Andrew Lanham, Yipeng Wang, Meizhi Wang, Sirish Oruganti, Jaydeep P Kulkarni

University of Texas at Austin, Circuit Research Lab

Paper 29.2, Demo Session 2

The University of Texas at Austin
Chandra Department of Electrical and Computer Engineering
Cockrell School of Engineering

Snap-SAT Design Demo Session (DS2)



The design was demonstrated last night on Demo Session 2. 😀

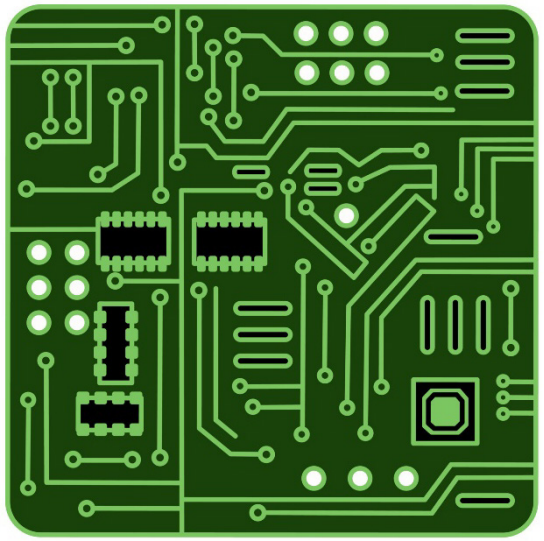
Outline

- **Motivation**
  - Background
  - Prior boolean **sat**isfiability problem (SAT) solvers
- **Proposed Snap-SAT Architecture**
  - Design highlights
  - SAT problem mapping example
  - Overall architecture
  - Circuit diagram
- **Silicon Prototype Measurements**
- **Snap-SAT Summary**

Outline

- **Motivation**
  - Background
    - Prior boolean **sat**isfiability problem (SAT) solvers
- **Proposed Snap-SAT Architecture**
  - Design highlights
  - SAT problem mapping example
  - Overall architecture
  - Circuit diagram
- **Silicon Prototype Measurements**
- **Snap-SAT Summary**

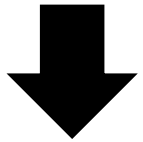
Boolean SATisfiability Problem Applications

Defense	Healthcare	Automation	Scheduling
			
Anti-aircraft mission planning	Vaccine development	Automatic test-pattern generation	Air-traffic control & taxi scheduling

Boolean Satisfiability Problem?

Boolean SATisfiability (SAT) Problem

Boolean equation in conjunctive normal form



$$F(x) = (\, x_0 \text{ OR } \overline{x_1} \text{ OR } x_5) \text{ AND } (x_1 \text{ OR } \overline{x_3} \text{ OR } \overline{x_4})$$

Note: OR = disjunction;
AND = conjunction;

Boolean SATisfiability (SAT) Problem

Clause = disjunction between *k* literals

e.g. *3-SAT: 3 literals in every clause*

4-SAT: 4 literals in every clause

$$F(x) = (\, x_0 \text{ OR } \overline{x_1} \text{ OR } x_5) \text{ AND } (x_1 \text{ OR } \overline{x_3} \text{ OR } \overline{x_4})$$

Boolean variables: $(x_0, x_1, x_2 \ldots)$

Literal = a variable/its negation

Boolean SATisfiability (SAT) Problem

- Objective:
  - Find values of *n* Boolean variables ($x_i \in \{0,1\}^n$) that **satisfy** all clauses in a Boolean formula (*F(x)*).

Clause

$$F(x) = (\, x_0 \text{ OR } \overline{x_1} \text{ OR } x_5) \text{ AND } (x_1 \text{ OR } \overline{x_3} \text{ OR } \overline{x_4}) = \textbf{TRUE}$$

Boolean variables

$(x_0, x_1, x_2 \ldots)$

Boolean SATisfiability (SAT) Problem

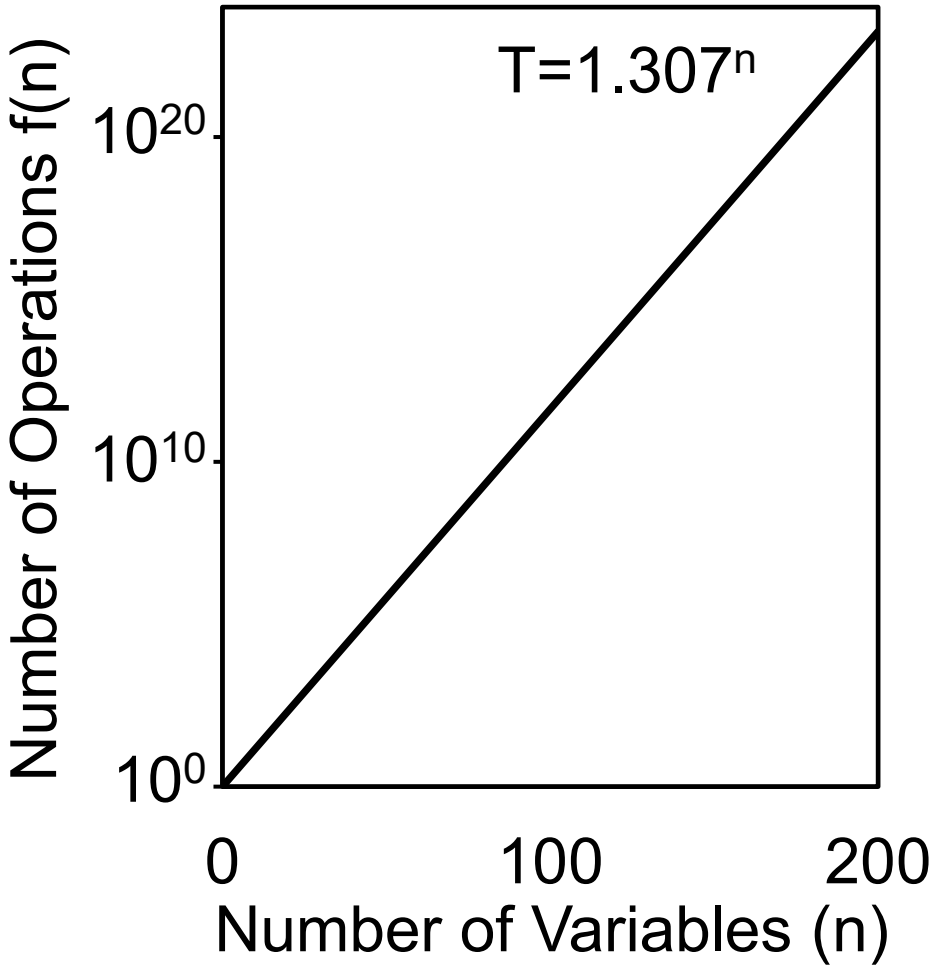
- Objective:
  - Find values of *n* Boolean variables ($x_i \in \{0,1\}^n$) that **satisfy** all clauses in a Boolean formula (*F(x)*).

$$F(x) = (\, x_0 \text{ OR } \overline{x_1} \text{ OR } x_5) \text{ AND } (x_1 \text{ OR } \overline{x_3} \text{ OR } \overline{x_4}) \textbf{ = TRUE}$$

$x_0 = 1$ $\qquad$ $x_3 = 0$

Motivation for SAT Solver Hardware Solution

- Number of variables ⬆
- # of operations ⬆ exponentially ☹
- Intensive Data:
  - ☹ Data movement ⬆
  - ☹ Computational latency ⬆
  - ☹ Computation energy cost ⬆



T.D. Hansen, '1.1 computing's energy problem (and what we can do about it).", ACM SIGACT Symposium

Hardware Accelerator?

for Boolean Satisfiability Problems

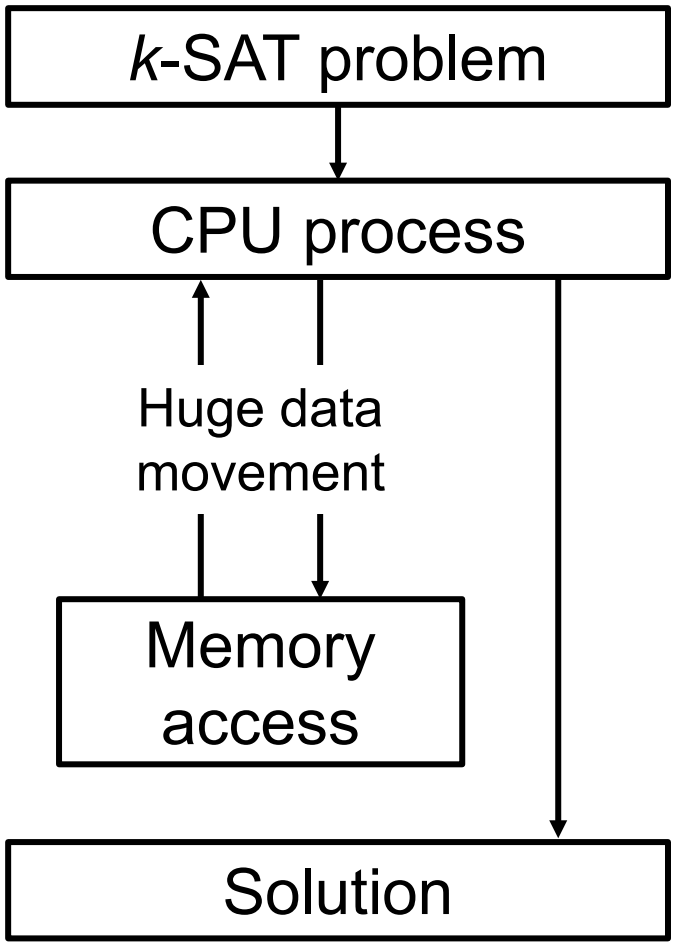
Outline

- **Motivation**
  - Background
  - Prior boolean **sat**isfiability problem (SAT) solvers
- **Proposed Snap-SAT Architecture**
  - Design highlights
  - SAT problem mapping example
  - Overall architecture
  - Circuit diagram
- **Silicon Prototype Measurements**
- **Snap-SAT Summary**

Software SAT Solver: MiniSAT

- Features:
  - Software algorithm
  - 😊 Low implement cost
  - 😊 Scalable to any problem size
- Cons:
  - 😟 Process on CPU
  - 😟 Energy cost (10^{13} times 32b integer addition energy (45nm))
  - 😟 Huge data movement
  - 😟 Frequent memory access

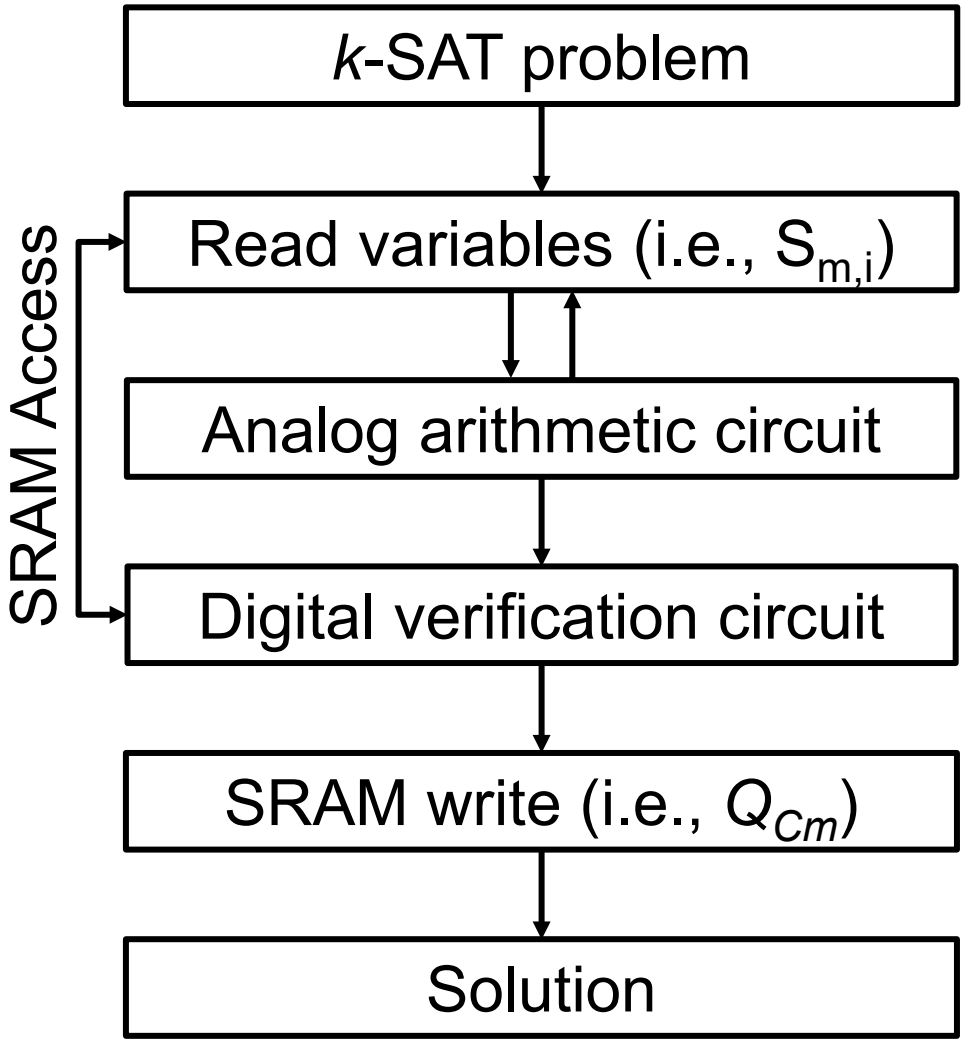
M. Horowitz, ISSCC, 2014



N. Eén, Springer, pp. 502-518, 2003

Analog SAT Solver – AC-SAT

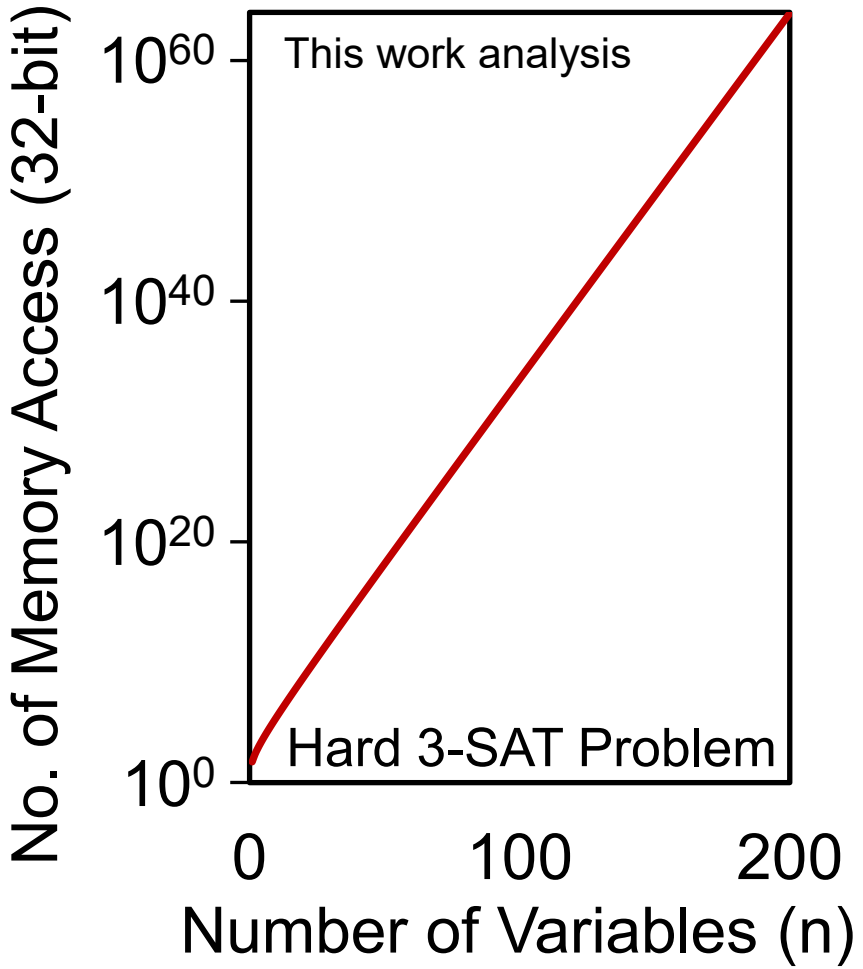
- Features:
  - Mix-signal design
  - Based on the deterministic continuous-time dynamical system
  - 😊 Innovative design
- Cons:
  - 😟 Flexibility
  - 😟 Scalability
  - 😟 Frequent SRAM access

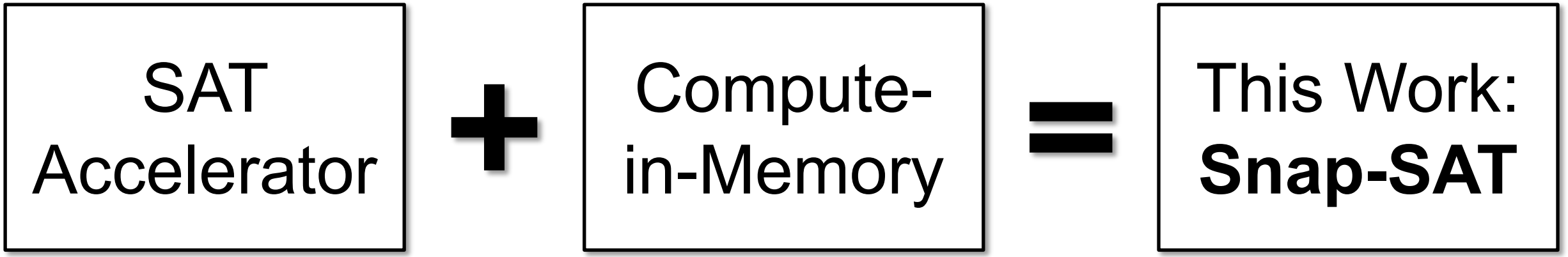


M. Chang, CICC, 2022

Compute-in-Memory (CIM) Motivations

- Variables are stored in memory
- SAT algorithms are CIM-friendly
  - Iterative computations
  - 1-bit data flips of variables
  - Local write back
- CIM design considerations
  - Reconfigurability
  - Scalability
  - Flexibility
  - Reliability

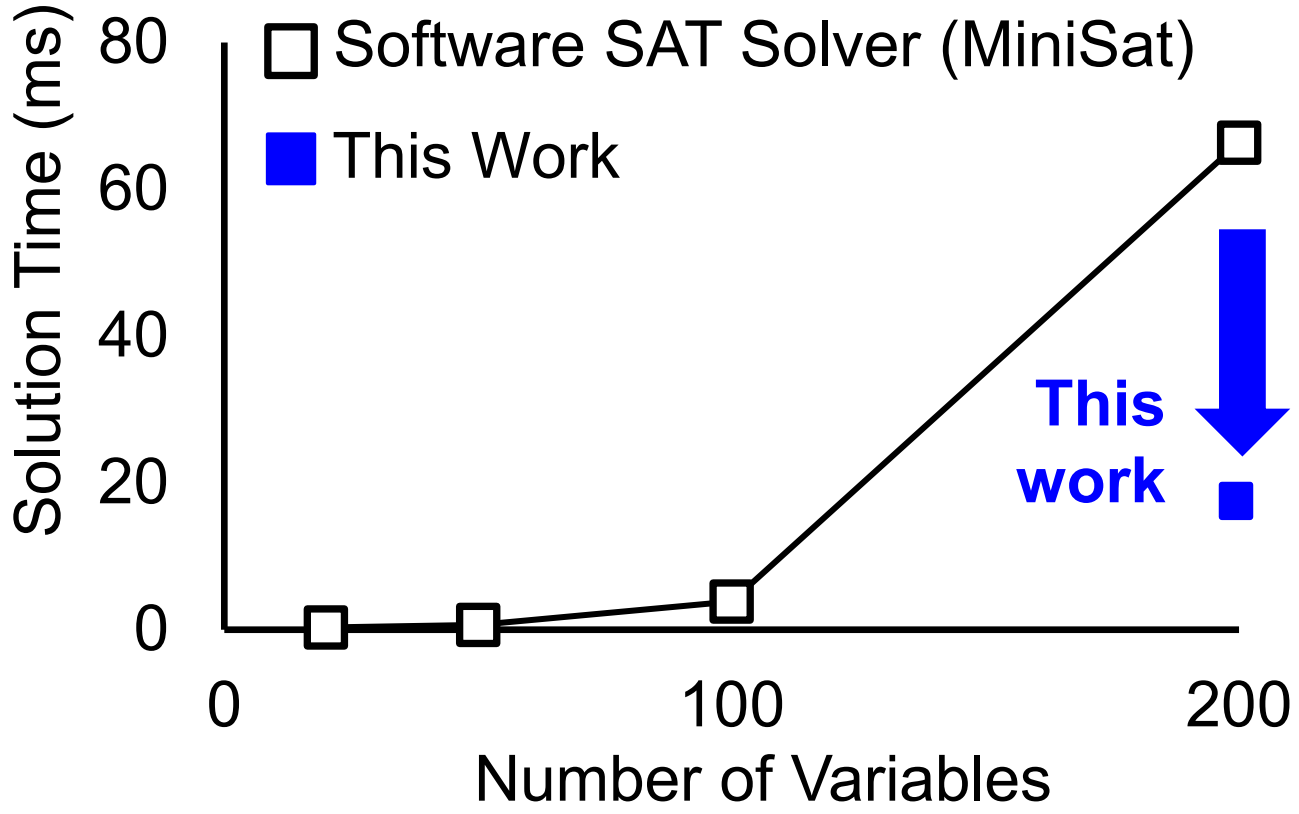




Outline

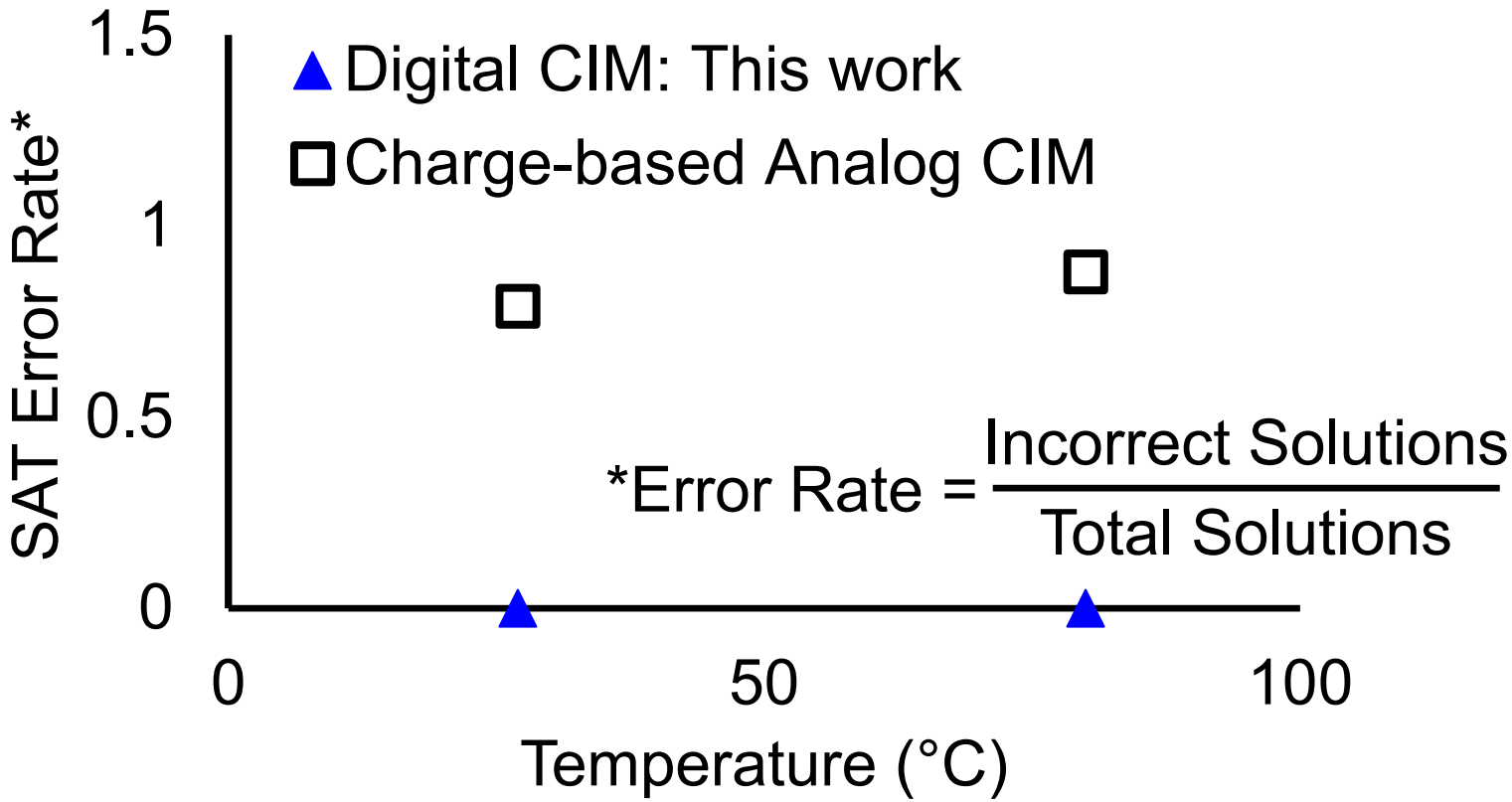
- Motivation
  - Background
  - Prior boolean satisfiability problem (SAT) solvers
- **Proposed Snap-SAT Architecture**
  - Design highlights
  - SAT problem mapping example
  - Overall architecture
  - Circuit diagram
- Silicon Prototype Measurements
- Snap-SAT Summary

Snap-SAT Design Highlights: High Performance



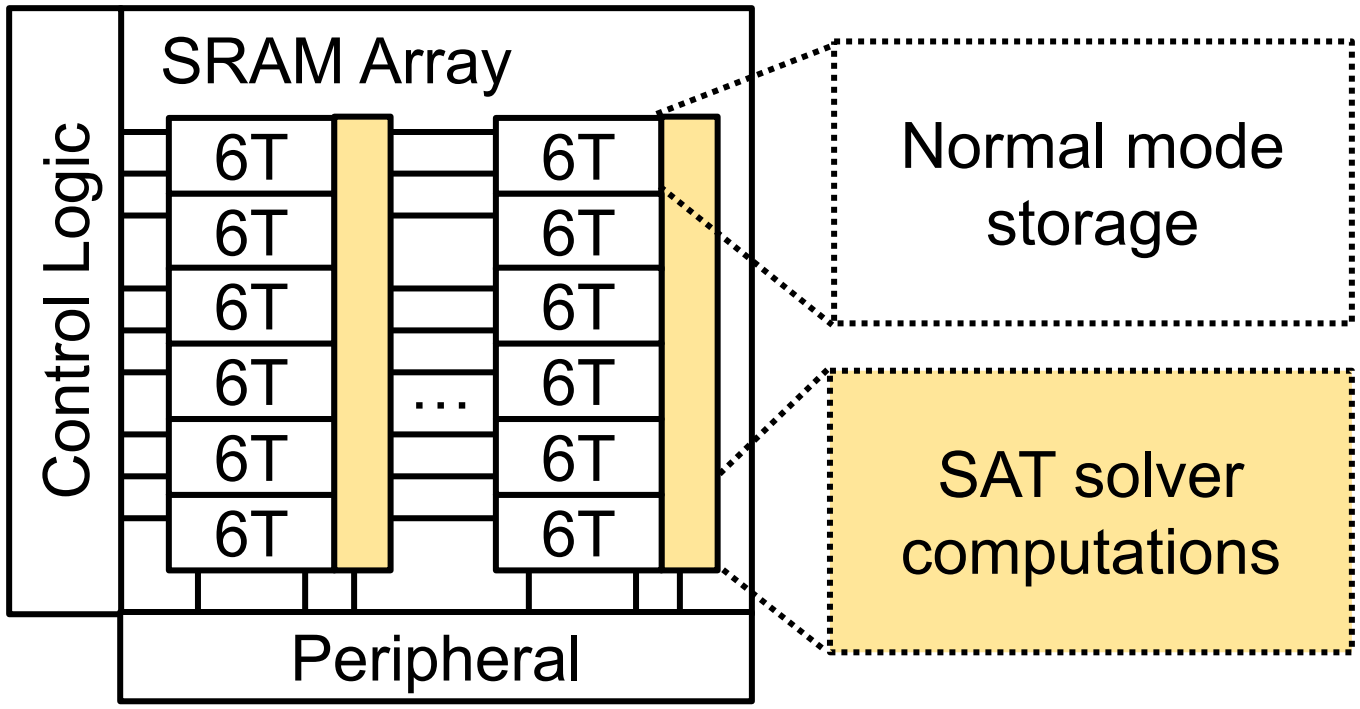
- **Snap-SAT**: All clauses are computed in one-shot
- **Snap-SAT**: Massive parallel in-memory computations
- Clauses/variables are stationary: no intra-memory or off-chip data movement

Snap-SAT Design Highlights: Computation Accuracy



- Accurate all-digital computations across temperature range
- Scalable to other CMOS technology nodes
- No data converter/charge share is needed for SAT computations

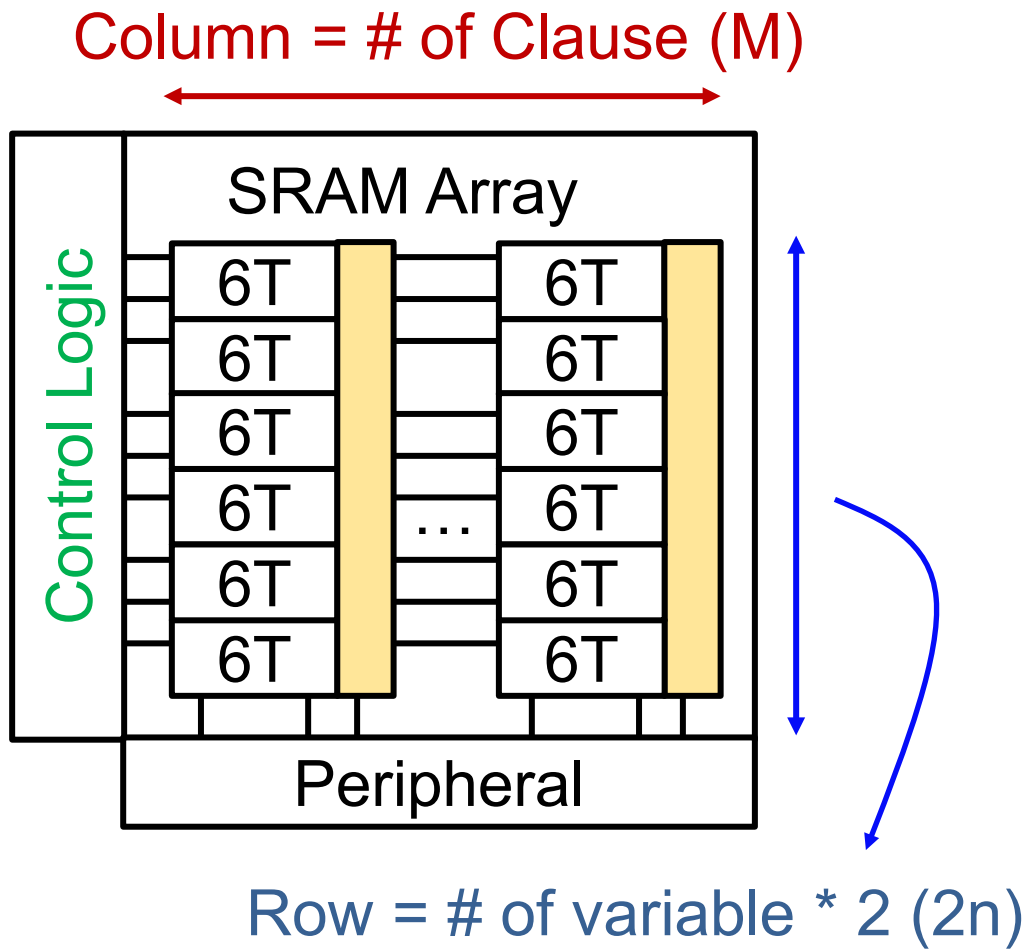
Snap-SAT Design Highlights: Reconfigurability



- Reuse existing SRAM array with minimal modifications for SAT computations
- Minimize area overhead of SAT specific in-memory computation circuits
- SRAM array can be used as regular memory in normal mode operations

Snap-SAT Design Highlights: Scalability

- Scalable w/ row size
  - *k* parameter in *k*-SAT
  - Number of variables (n)
- Scalable w/ column size
  - Number of clauses (M)
- Flexible with control logic
  - SAT algorithms
- Flexible for large-scale hard problems



Outline

- Motivation
  - Background
  - Prior boolean satisfiability problem (SAT) solvers
- **Proposed Snap-SAT Architecture**
  - Design highlights
  - SAT problem mapping example
  - Overall architecture
  - Circuit diagram
- Silicon Prototype Measurements
- Snap-SAT Summary

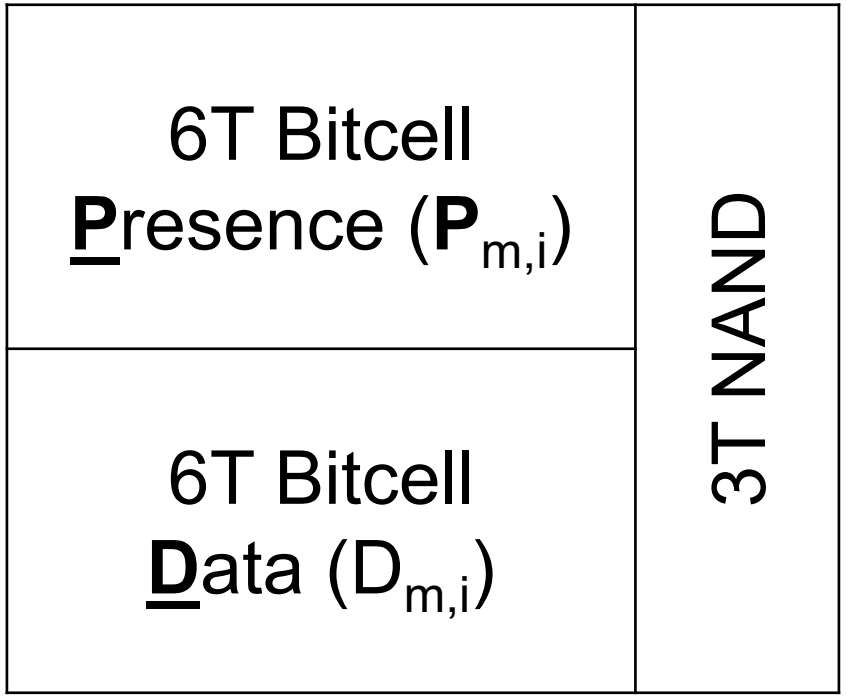
SRAM Array Mapping

$$F(x) = (x_0 \text{ OR } \overline{x_1} \text{ OR } x_5) \text{ AND } (x_1 \text{ OR } \overline{x_3} \text{ OR } \overline{x_4})$$

SRAM Array Mapping

$$F(x) = (x_0 \text{ OR } \overline{x_1} \text{ OR } x_5) \text{ AND } (x_1 \text{ OR } \overline{x_3} \text{ OR } \overline{x_4})$$

Dual SRAM Bitcells with 3T NAND



SRAM Array Mapping

$$F(x) = (x_0 \text{ OR } \overline{x_1} \text{ OR } x_5) \text{ AND } (x_1 \text{ OR } \overline{x_3} \text{ OR } \overline{x_4})$$

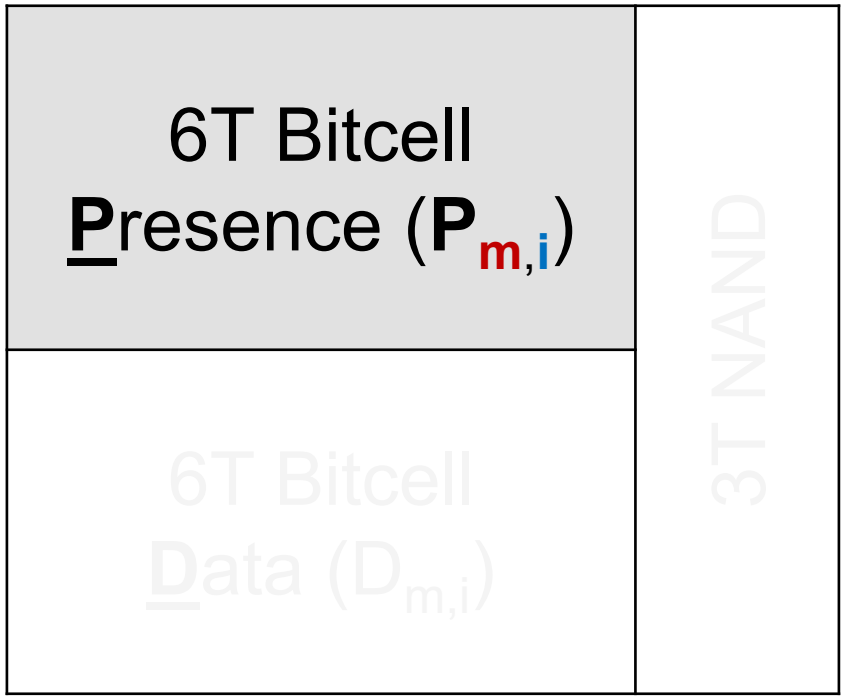
Presence:

*Whether variable **i** (x_i) or its negation ($\overline{x_i}$) is present in clause **m** or not*

$$\mathbf{P_{m,i}} = \begin{cases} 1, \ x_i \text{ or } \overline{x_i} \text{ in clause m} \\ 0, \text{ no } x_i \text{ or } \overline{x_i} \text{ in clause m} \end{cases}$$

Each variable has different value of P for each clause

Dual SRAM Bitcells with 3T NAND



SRAM Array Mapping

$$F(x) = (x_0 \text{ OR } \overline{x_1} \text{ OR } x_5) \text{ AND } (x_1 \text{ OR } \overline{x_3} \text{ OR } \overline{x_4})$$

Data:

*Variable **i** data in clause **m***

Case 1:

x_i in the clause **m**

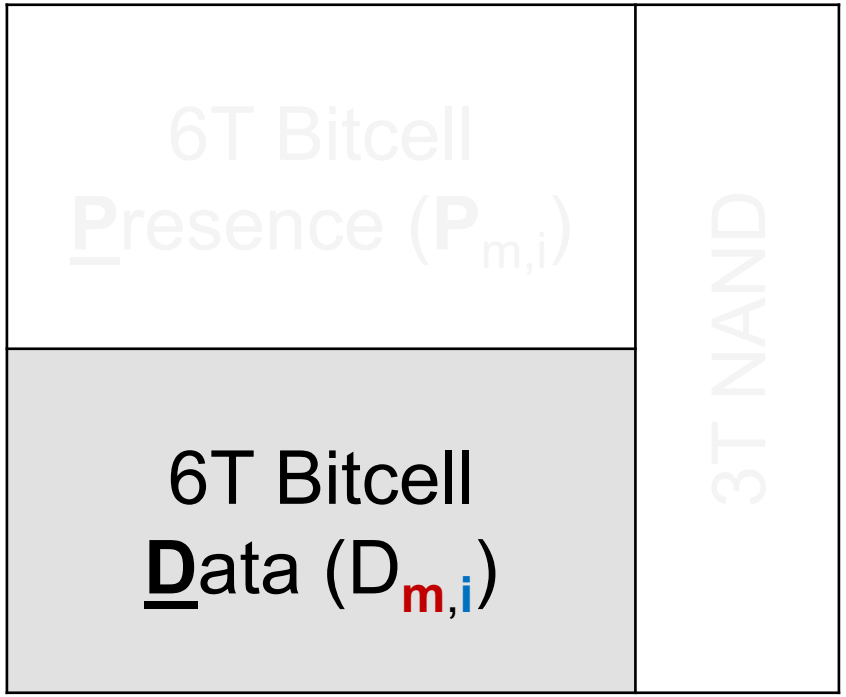
x_i	$D_{m,i}$
0	0
1	1

Case 2:

$\overline{x_i}$ in the clause **m**

x_i	$D_{m,i}$
0	1
1	0

Dual SRAM Bitcells with 3T NAND



SRAM Array Mapping

$F(x) = (x_0 \text{ OR } \overline{x_1} \text{ OR } x_5) \text{ AND}$

Clause 1 (C_1)

$(x_1 \text{ OR } \overline{x_3} \text{ OR } \overline{x_4})$

SRAM Column 1

x_1 is in the clause

$P_{1,1} = 1$

$D_{1,1} = x_1$

$P_{1,1}/D_{1,1}$: Clause 1; Variable 1

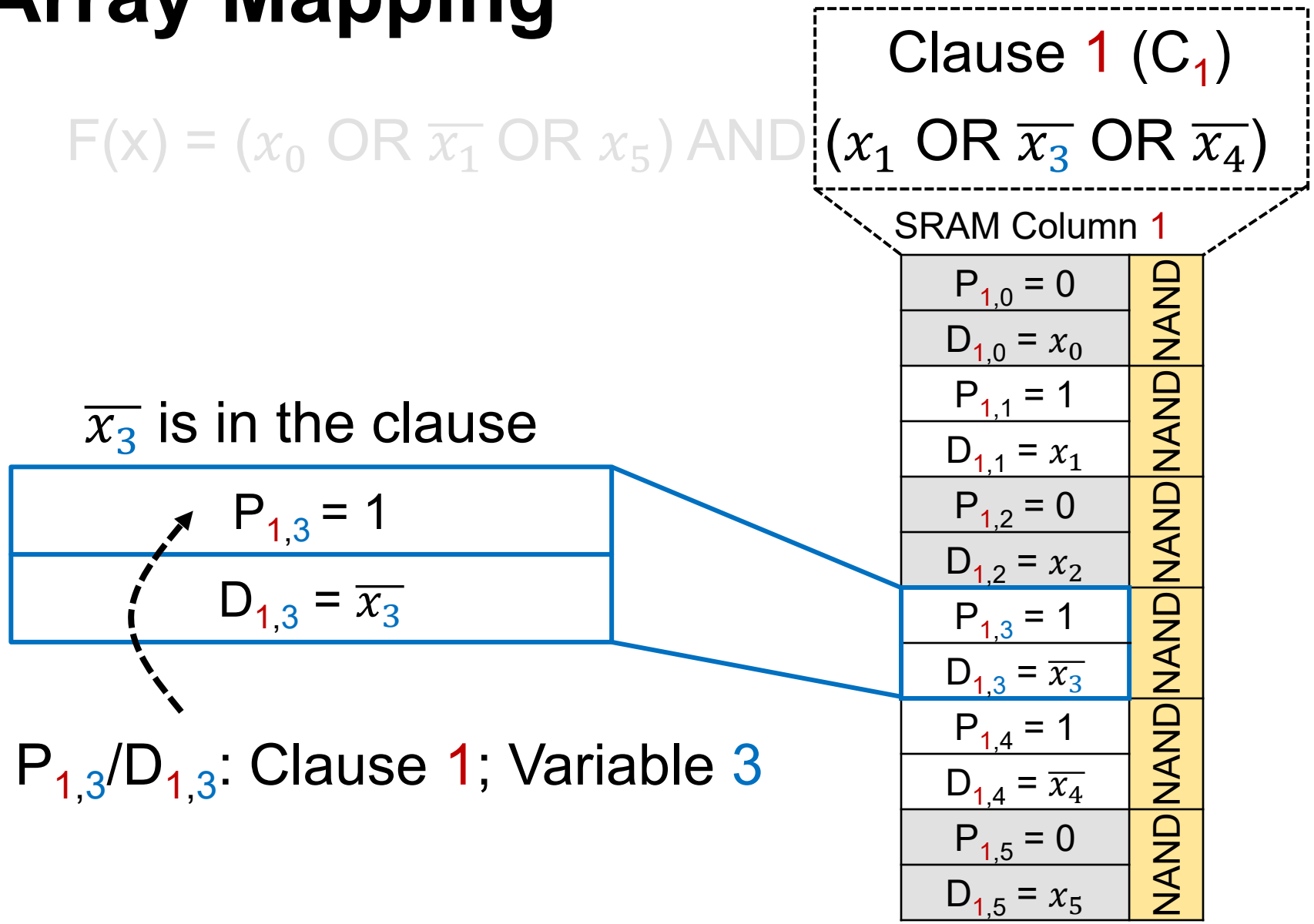
$P_{1,0} = 0$	NAND
$D_{1,0} = x_0$	
$P_{1,1} = 1$	NAND
$D_{1,1} = x_1$	
$P_{1,2} = 0$	NAND
$D_{1,2} = x_2$	
$P_{1,3} = 1$	NAND
$D_{1,3} = \overline{x_3}$	
$P_{1,4} = 1$	NAND
$D_{1,4} = \overline{x_4}$	
$P_{1,5} = 0$	NAND
$D_{1,5} = x_5$	

SRAM Array Mapping

$F(x) = (x_0 \text{ OR } \overline{x_1} \text{ OR } x_5) \text{ AND}$

Clause 1 (C_1)

$(x_1 \text{ OR } \overline{x_3} \text{ OR } \overline{x_4})$

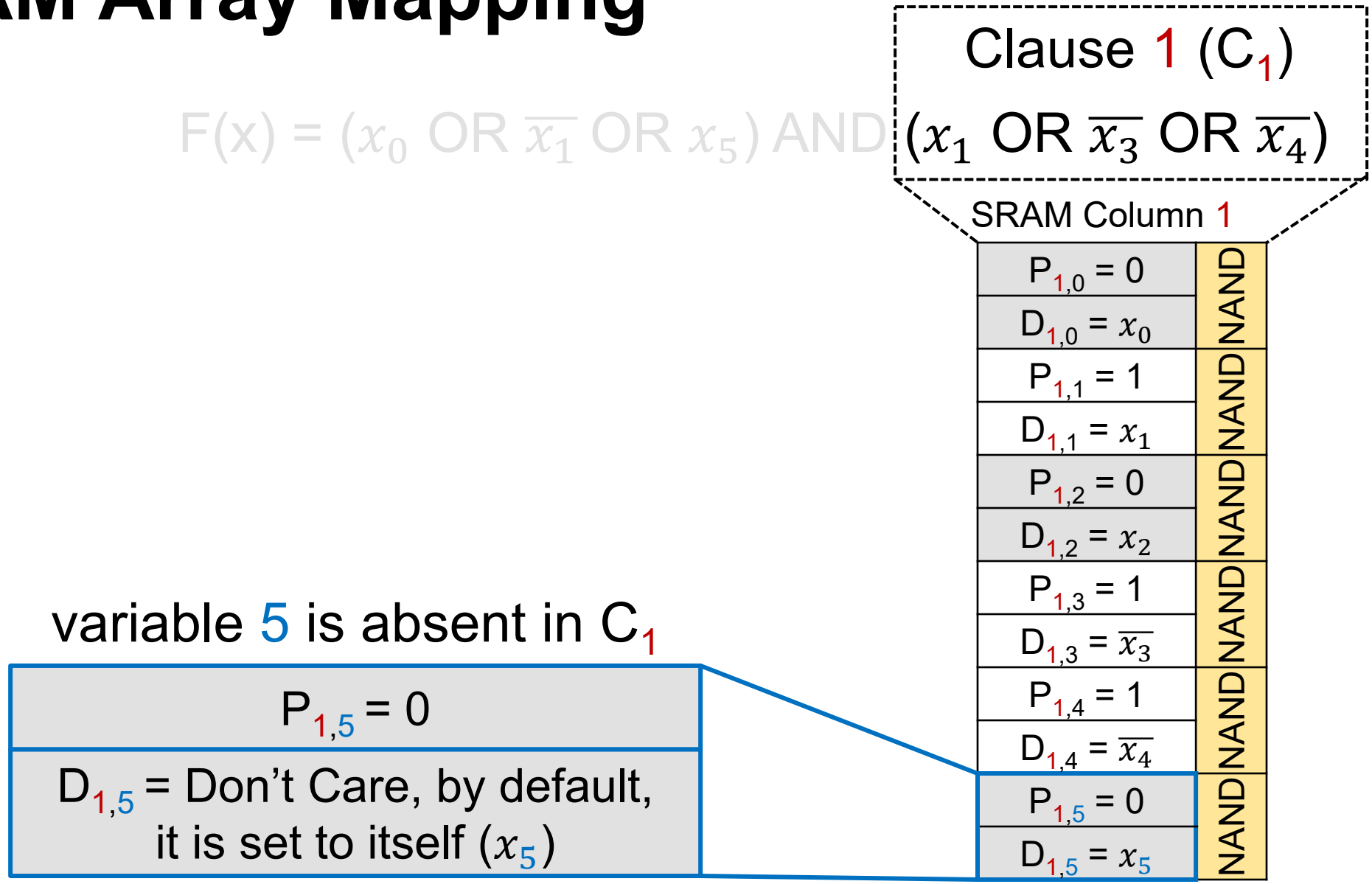


SRAM Array Mapping

$F(x) = (x_0 \text{ OR } \overline{x_1} \text{ OR } x_5) \text{ AND}$

Clause 1 (C_1)

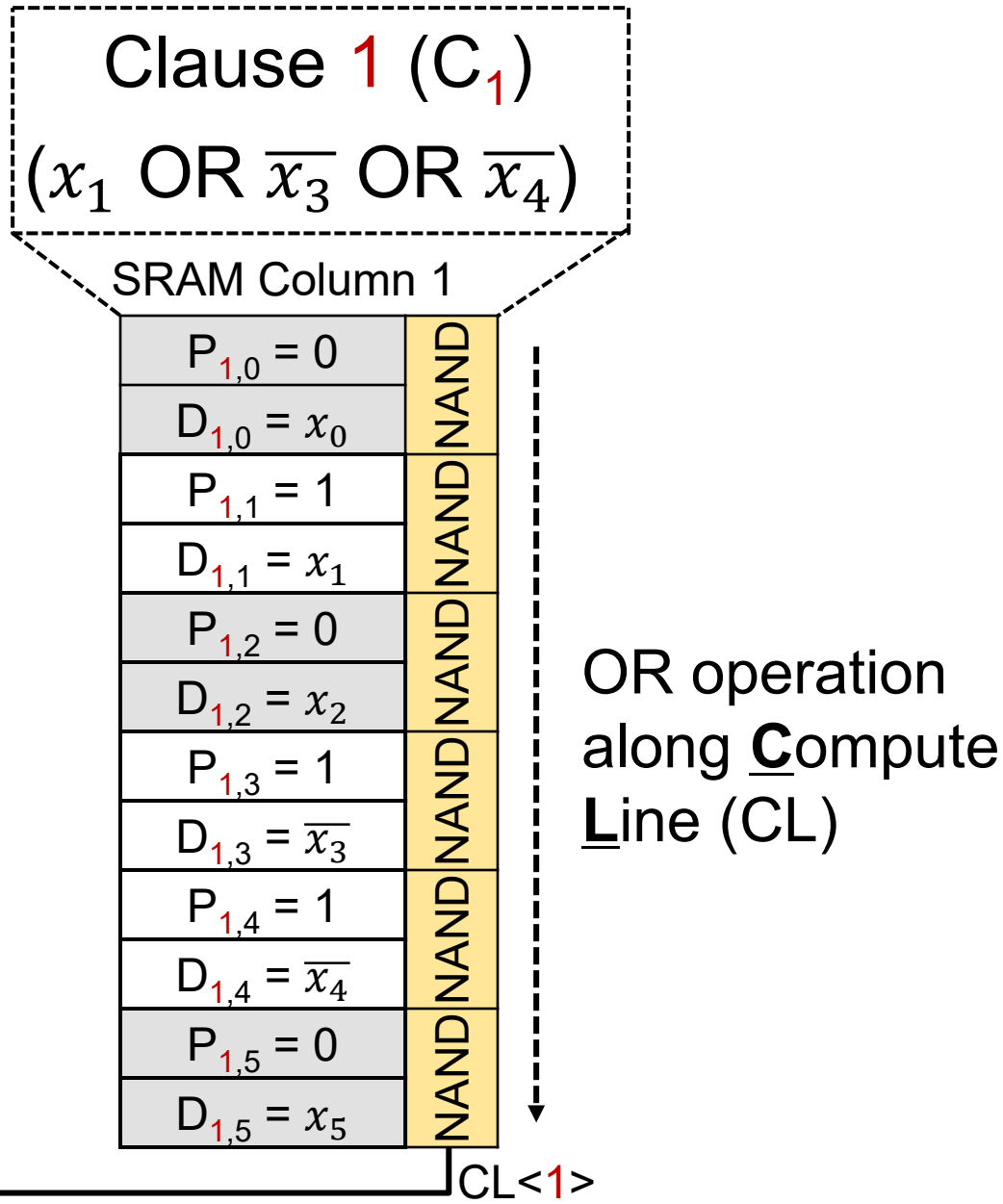
$(x_1 \text{ OR } \overline{x_3} \text{ OR } \overline{x_4})$



SRAM Array Mapping

F(x) = (x_0 OR $\overline{x_1}$ OR x_5) AND

- CL<1>: Clause 1 computation result
  - 1: Clause is unsatisfied (C_m=False)
  - 0: Clause is satisfied (C_m=True)
- Clause 1 is satisfied when
  - $x_1 = 1$
  - or $x_3 = 0$
  - or $x_4 = 0$

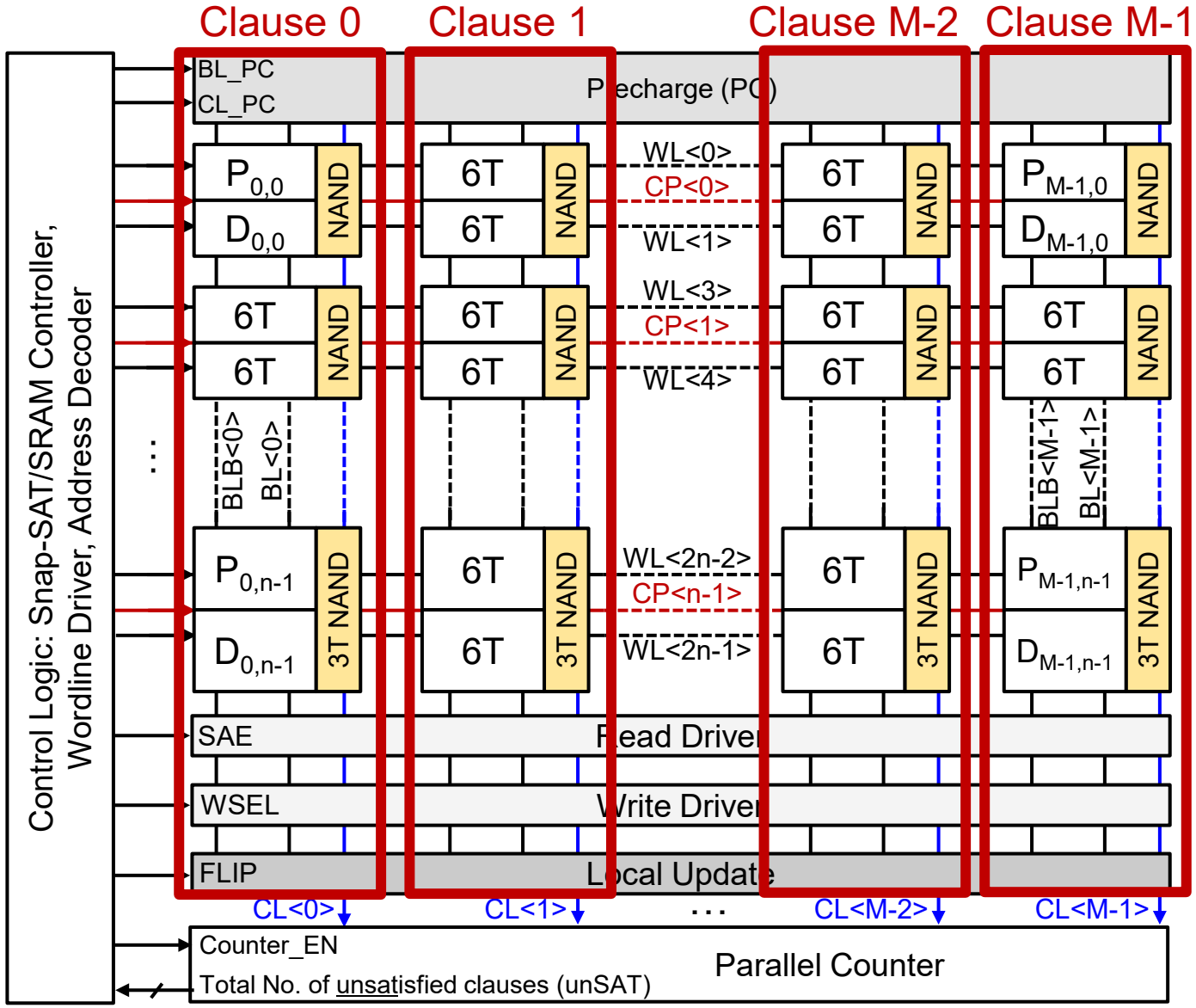


Outline

- Motivation
  - Background
  - Prior boolean satisfiability problem (SAT) solvers
- **Proposed Snap-SAT Architecture**
  - Design highlights
  - SAT problem mapping example
  - Overall architecture
  - Circuit diagram
- Silicon Prototype Measurements
- Snap-SAT Summary

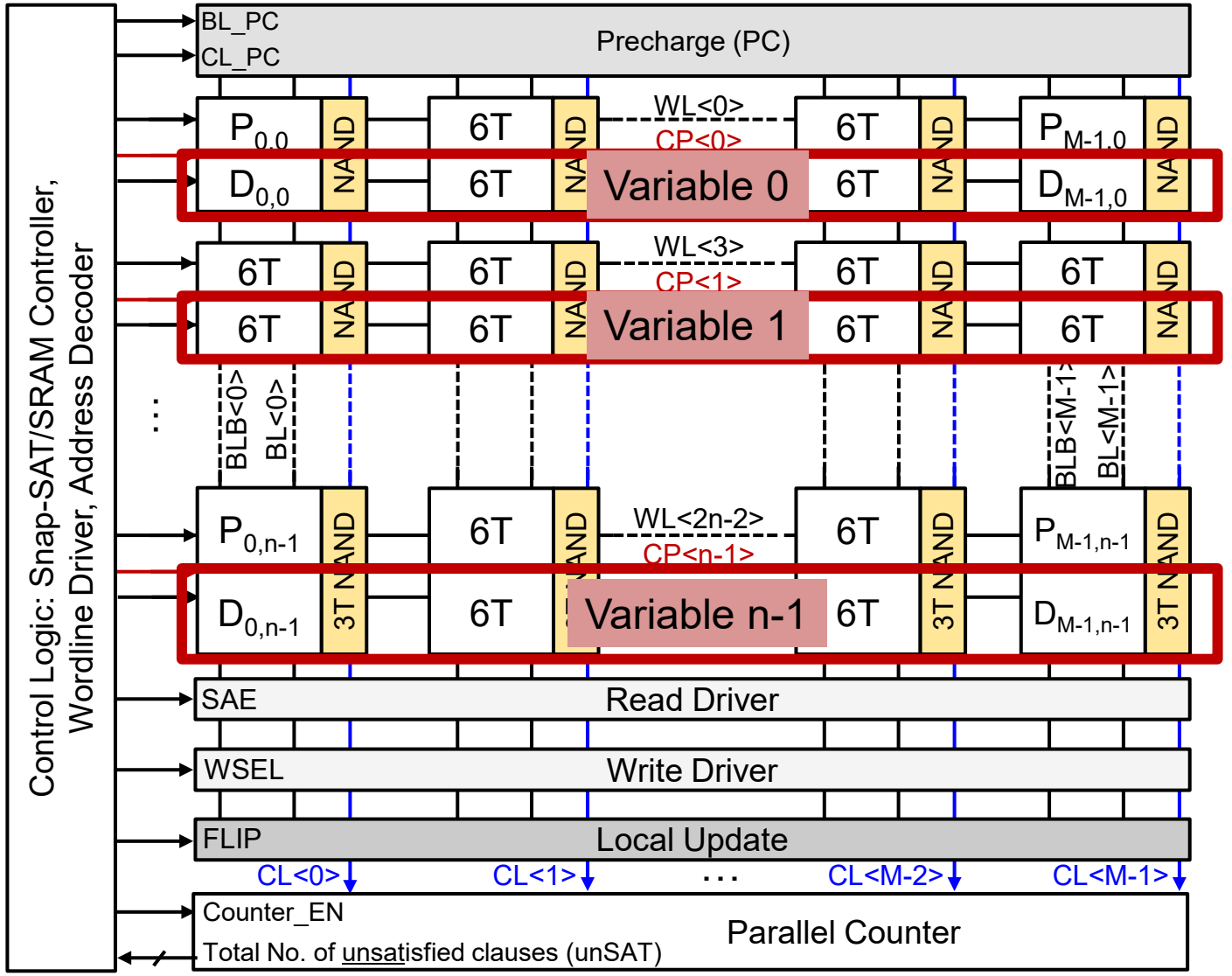
Snap-SAT Overall Architecture

- Each column represents a clause



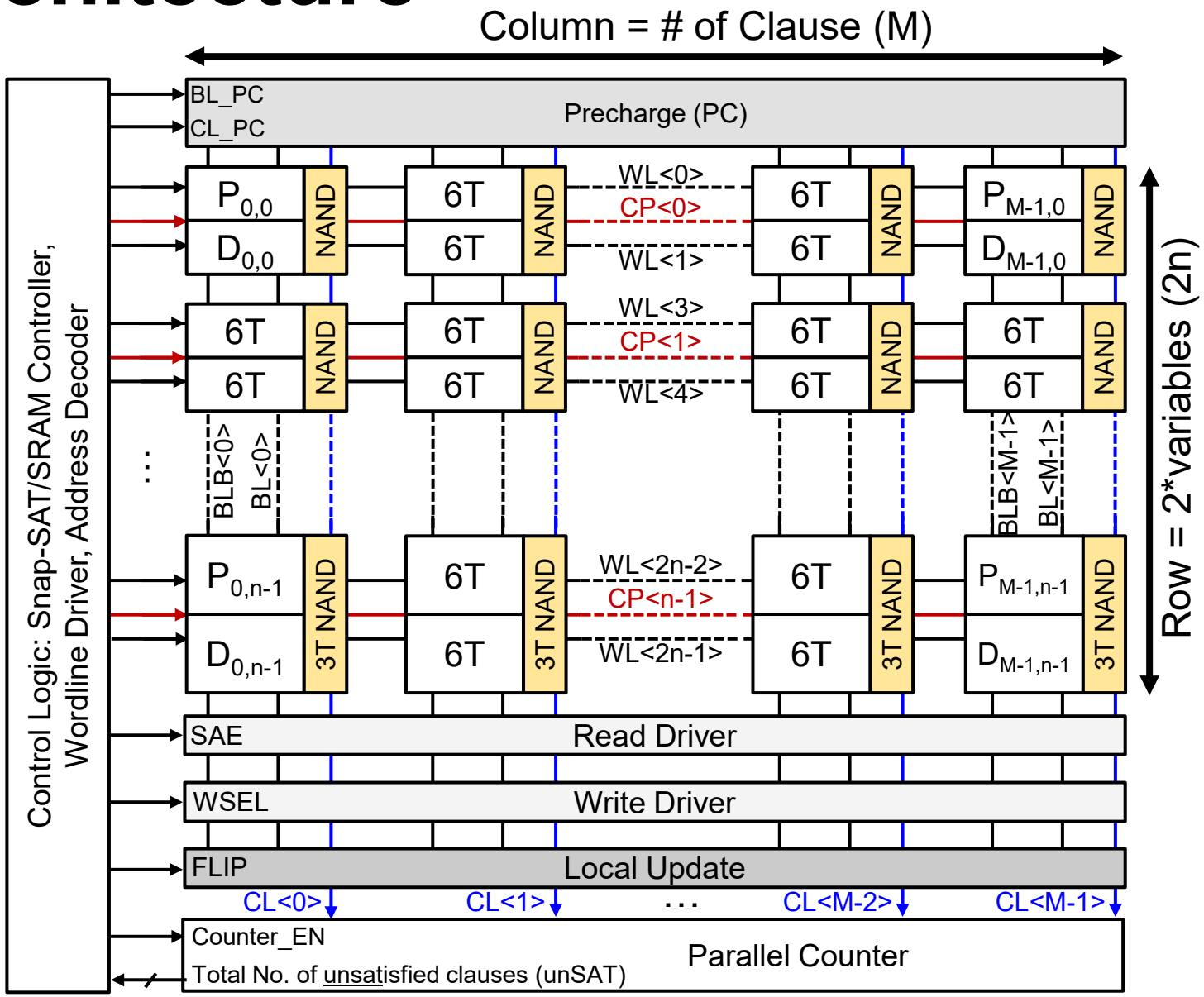
Snap-SAT Overall Architecture

- Each column represents a clause
- Each <u>data</u> row represents a variable
  - x_i or $\overline{x_i}$
  - Depends on SAT problem



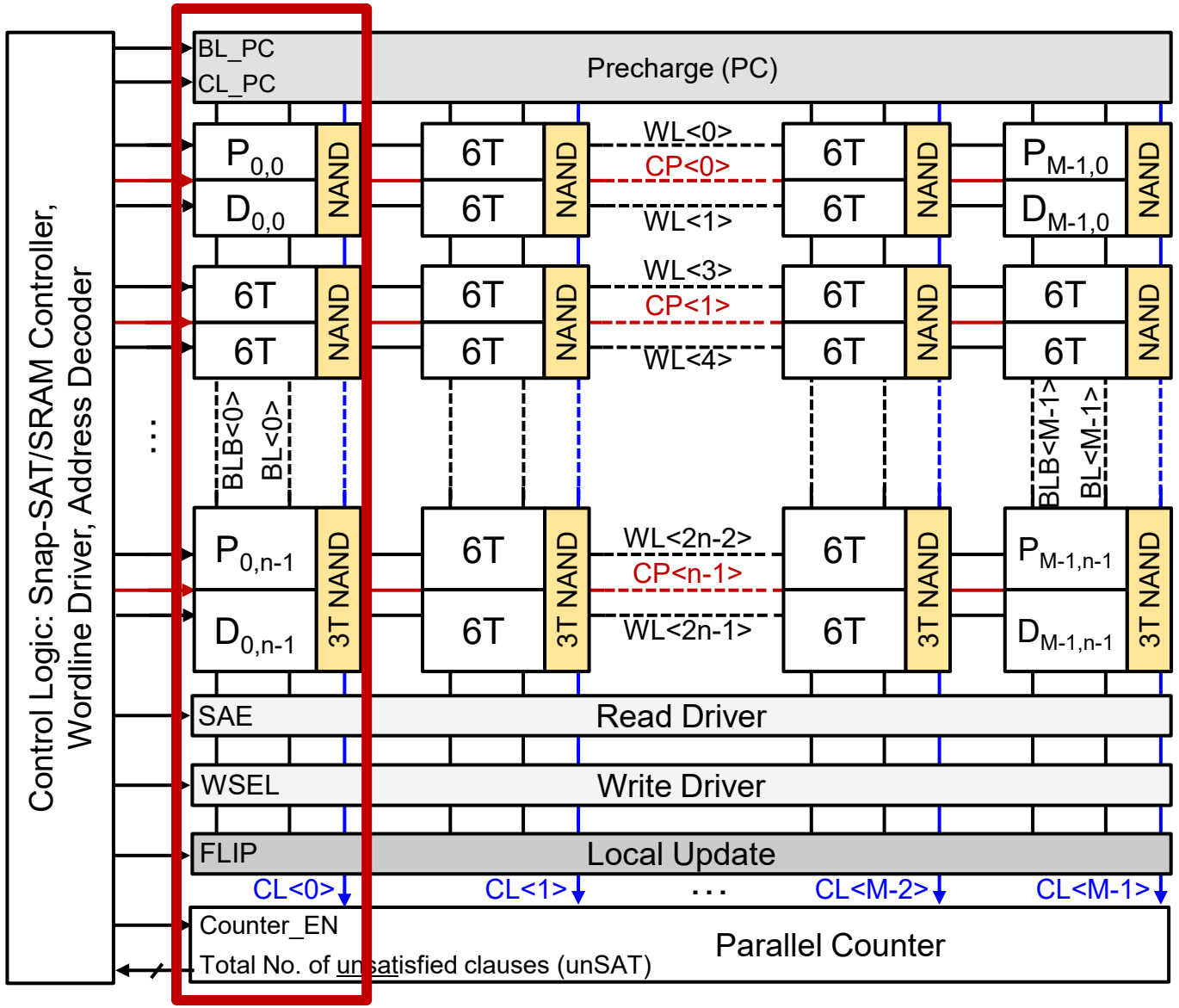
Snap-SAT Overall Architecture

- Column = # of clauses (M)
- Row = 2*variables (2n)
- Up to 128 variables
  - Prototype size restriction
- Up to 1024 clauses
  - Prototype size restriction
- With 131Kb SRAM



Snap-SAT Overall Architecture

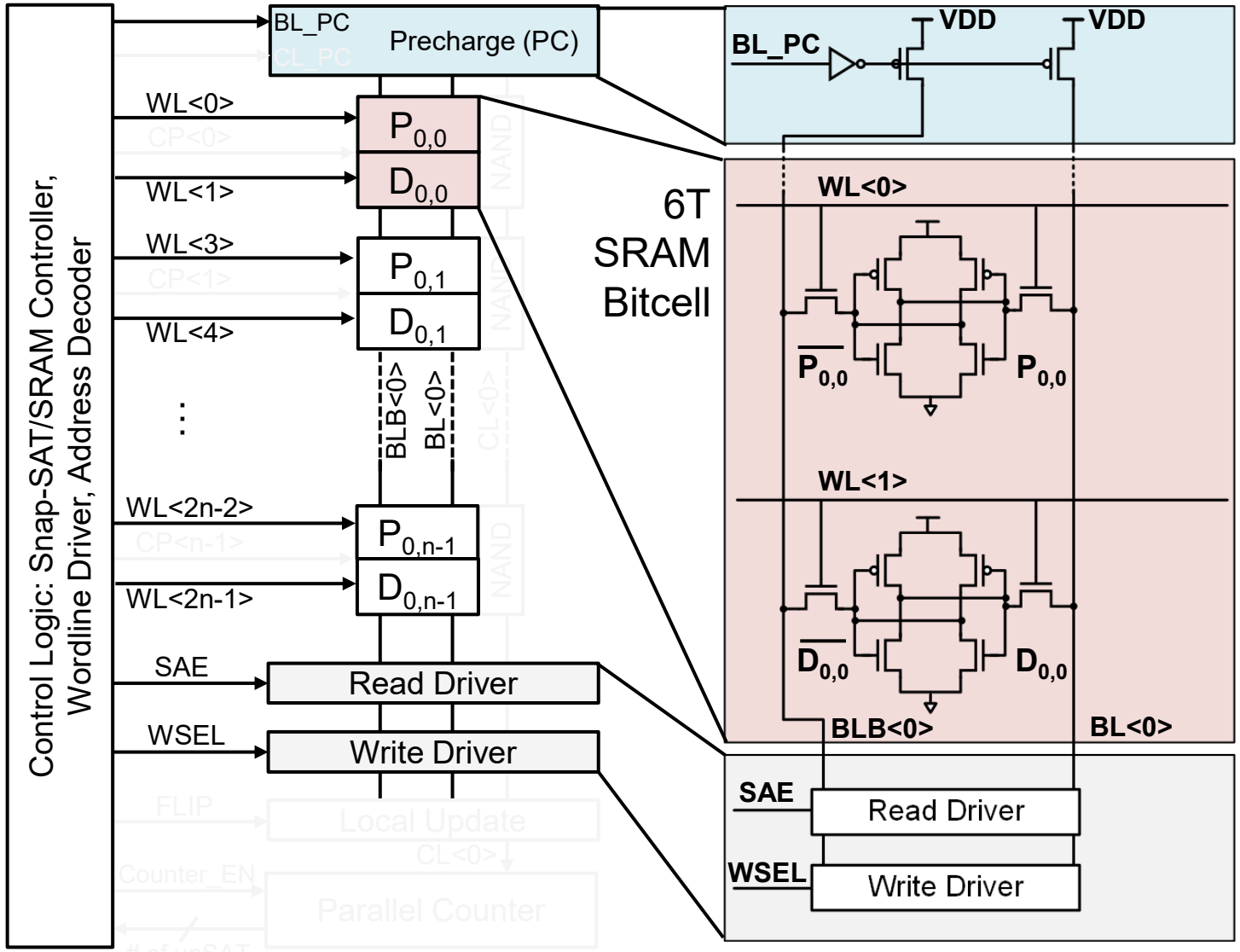
- Column = # of clauses (M)
- Row = 2*variables (2n)
- Up to 128 variables
  - Prototype size restriction
- Up to 1024 clauses
  - Prototype size restriction
- With 131Kb SRAM



Outline

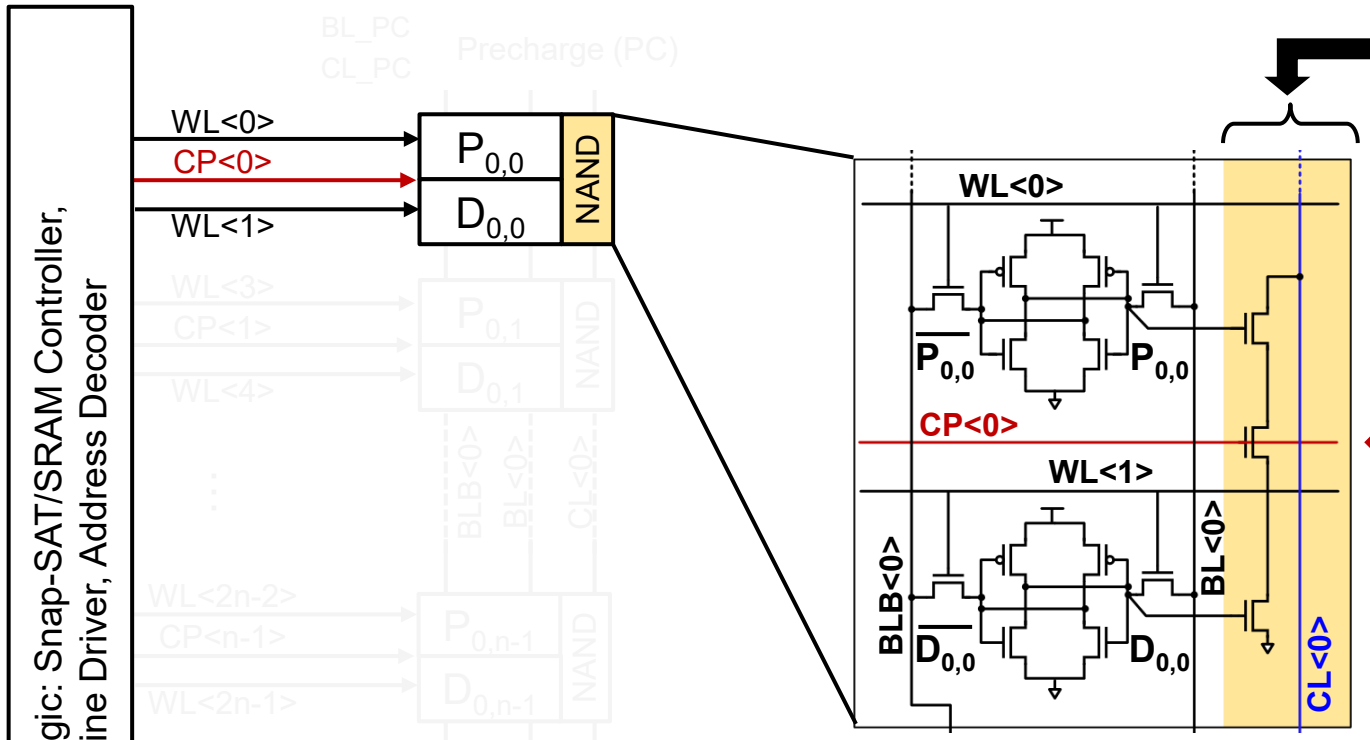
- Motivation
  - Background
  - Prior boolean satisfiability problem (SAT) solvers
- **Proposed Snap-SAT Architecture**
  - Design highlights
  - SAT problem mapping example
  - Overall architecture
  - Circuit diagram
- Silicon Prototype Measurements
- Snap-SAT Summary

Baseline SRAM Array Circuit



- Reuse existing 6T SRAM array
- Baseline SRAM circuit
  - Precharge
  - 6T bitcell
  - R/W driver
  - SRAM control logic
- Can be used as normal memory storage

Snap-SAT Circuit – 3T NAND Gate



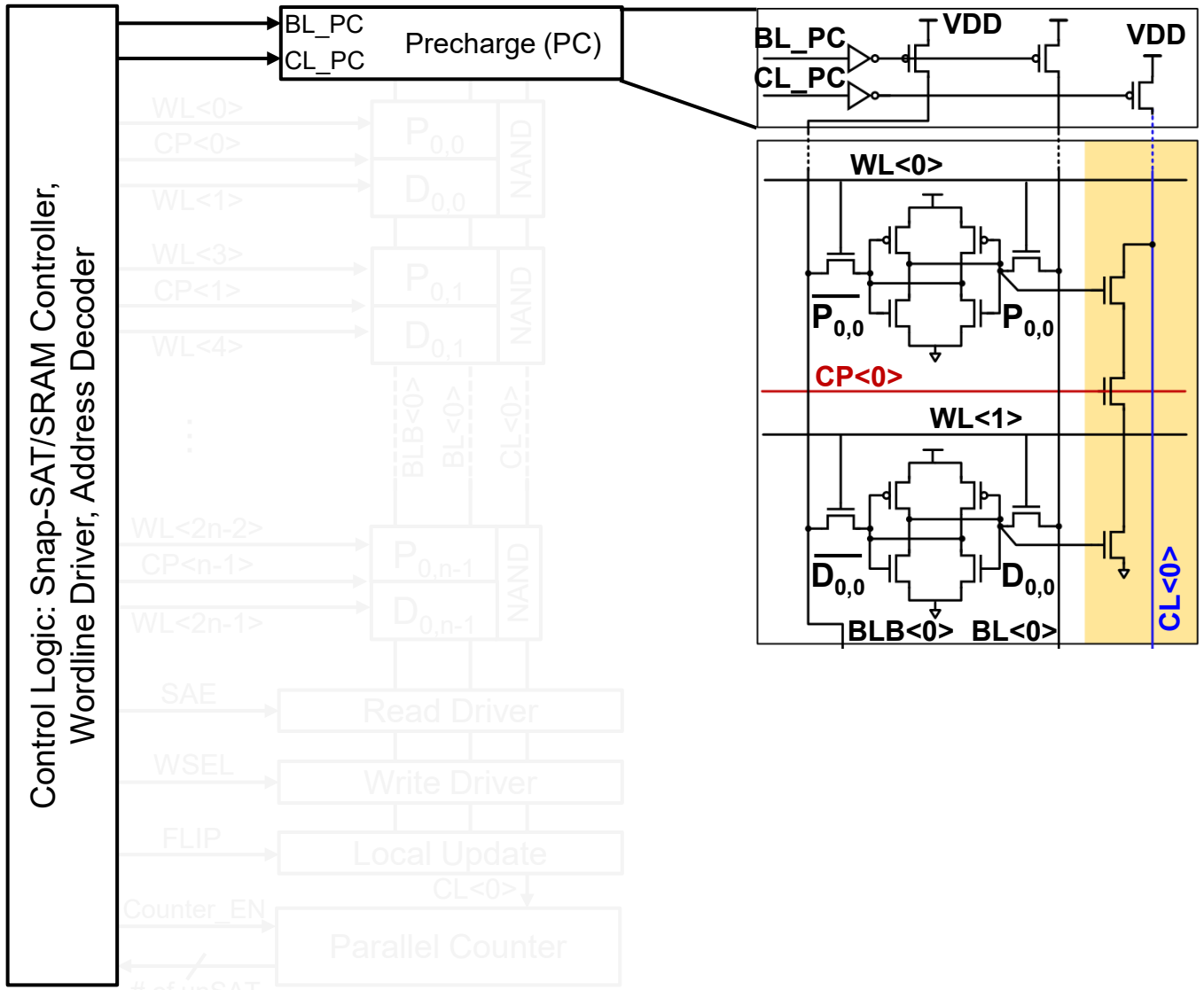
3-Transistor (3T) NAND Gate

Computation enable signal

- All CPs are enabled simultaneously
- One-shot clause computation

- Added 1.5 transistors per 6T bitcell
- Connect storage node to NAND gate inputs
- Clauses & variables are fully stationary

Snap-SAT Operation – Step 1: Precharge



1 Precharge

CL_PC

CL_PC turns ON

CP<n-1:0>

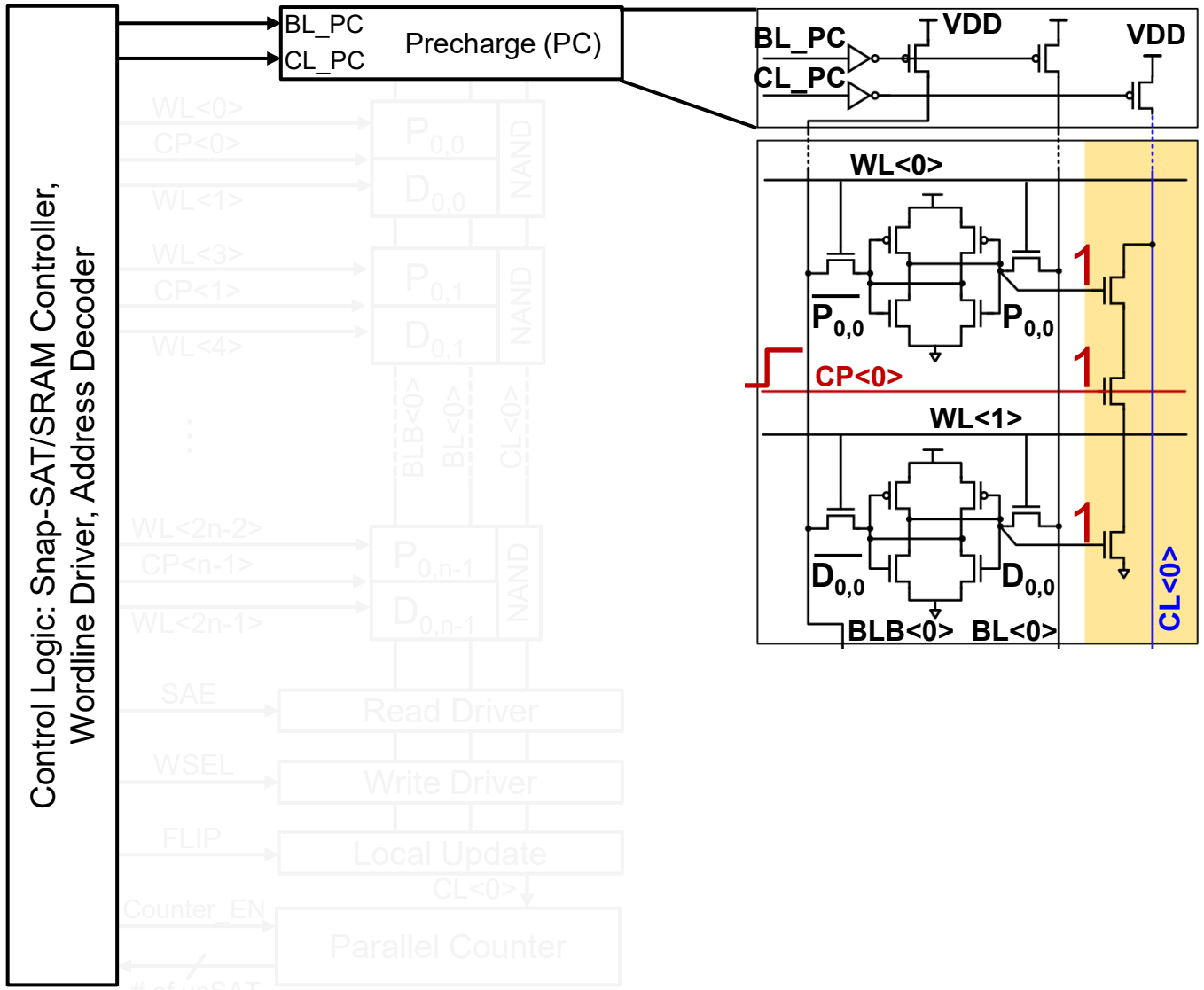
Compute Line (CL)

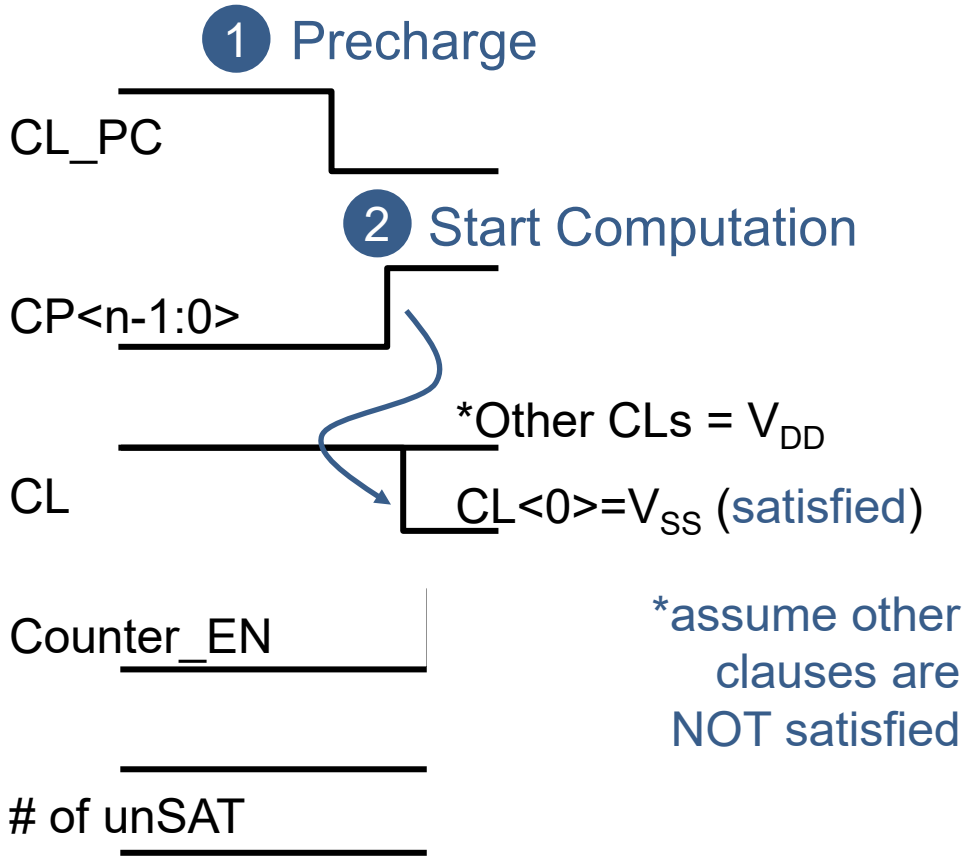
All CLs are precharged to V_{DD}

Counter_EN

# of unSAT

Snap-SAT Operation – Step 2: Compute

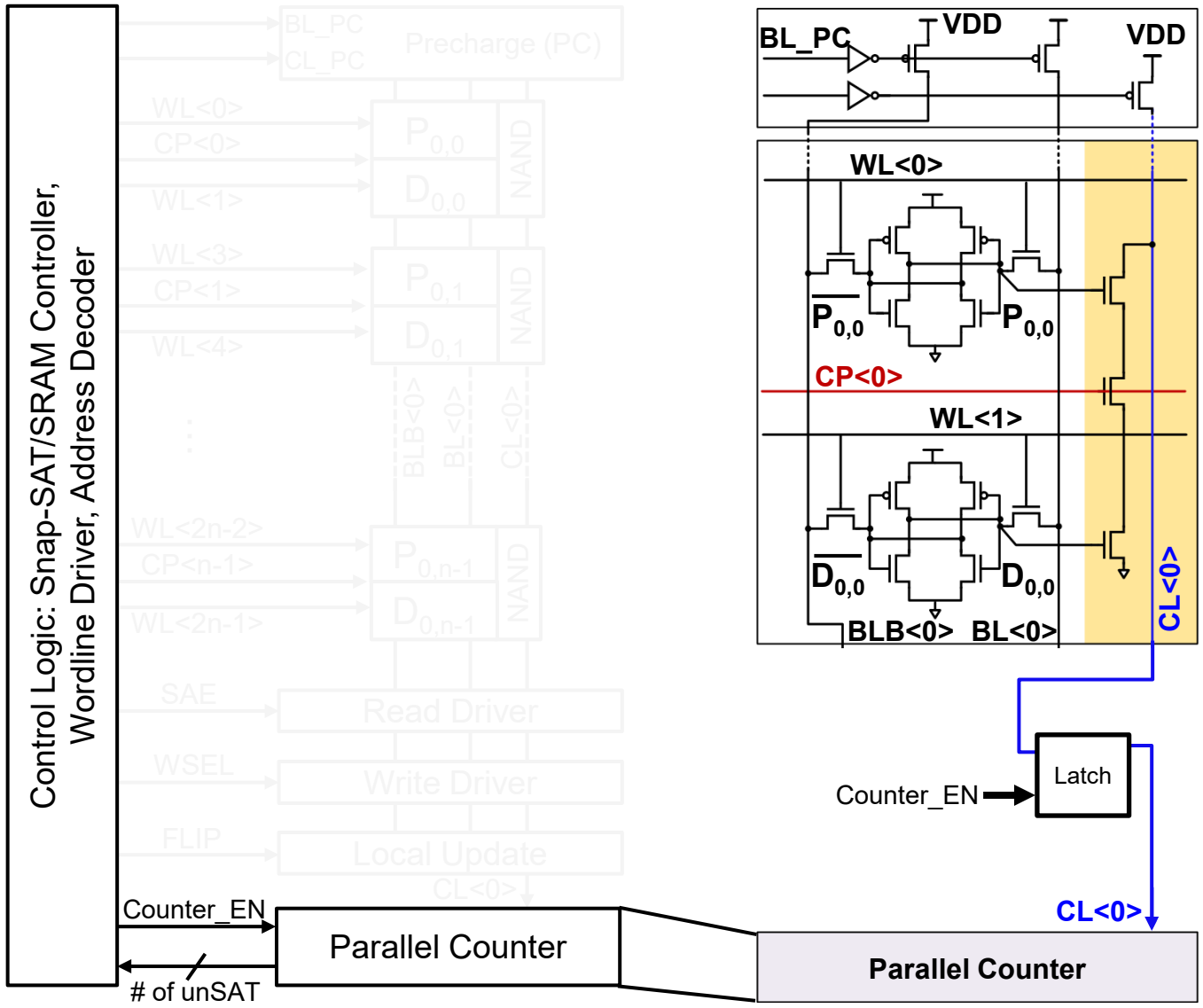


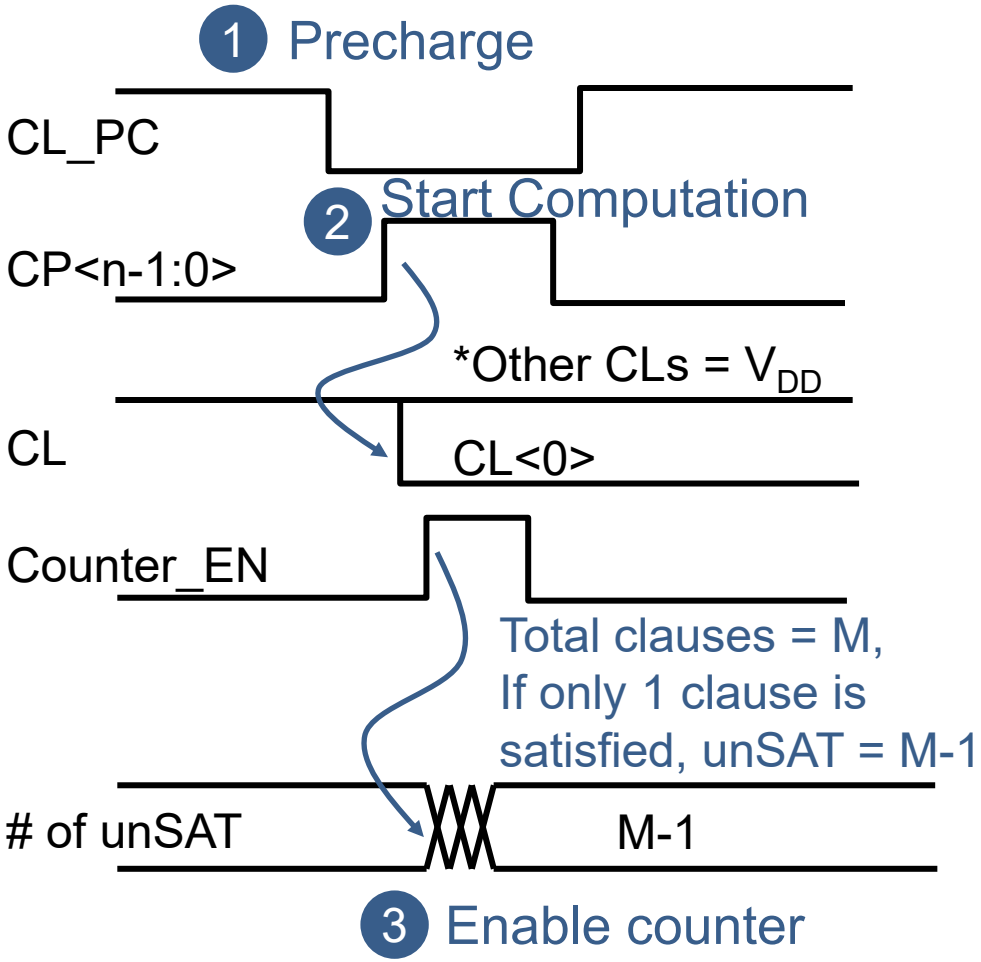


*assume other clauses are NOT satisfied

- All CPs are enabled simultaneously
- All clauses are computed in parallel
- OR operation along CL

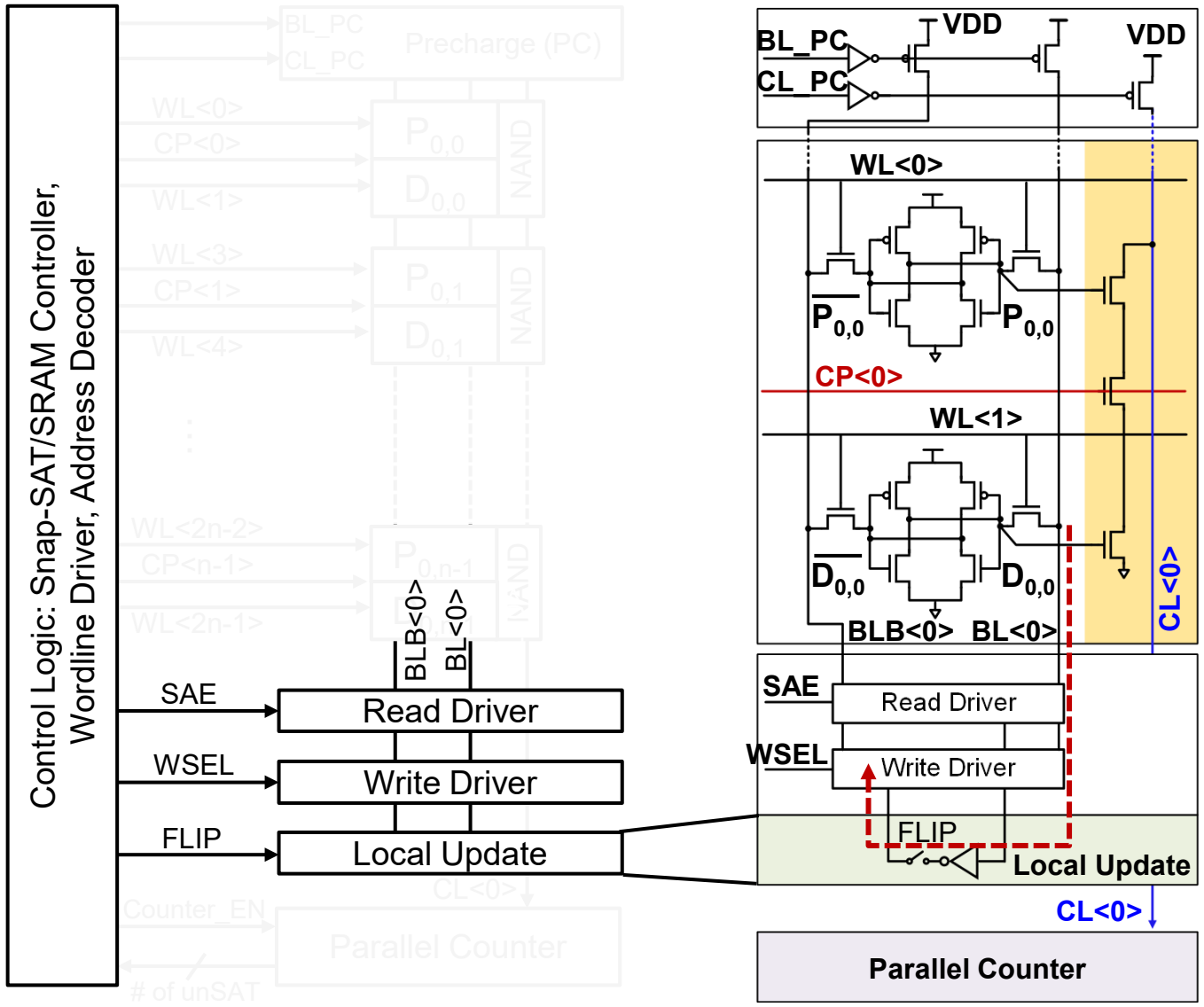
Snap-SAT Operation – Step 3: Counter

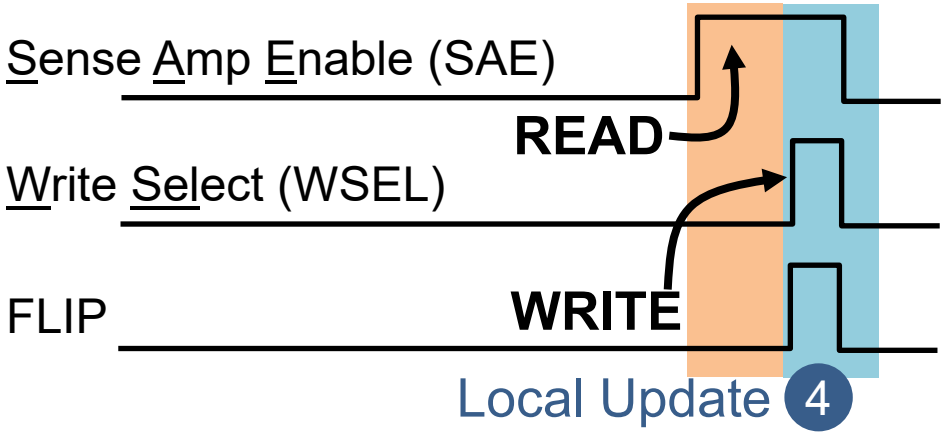




- Counter sums up all the CL results
- **Counter output = # of unsatisfied clauses**

Snap-SAT Operation – Step 4: Local Update



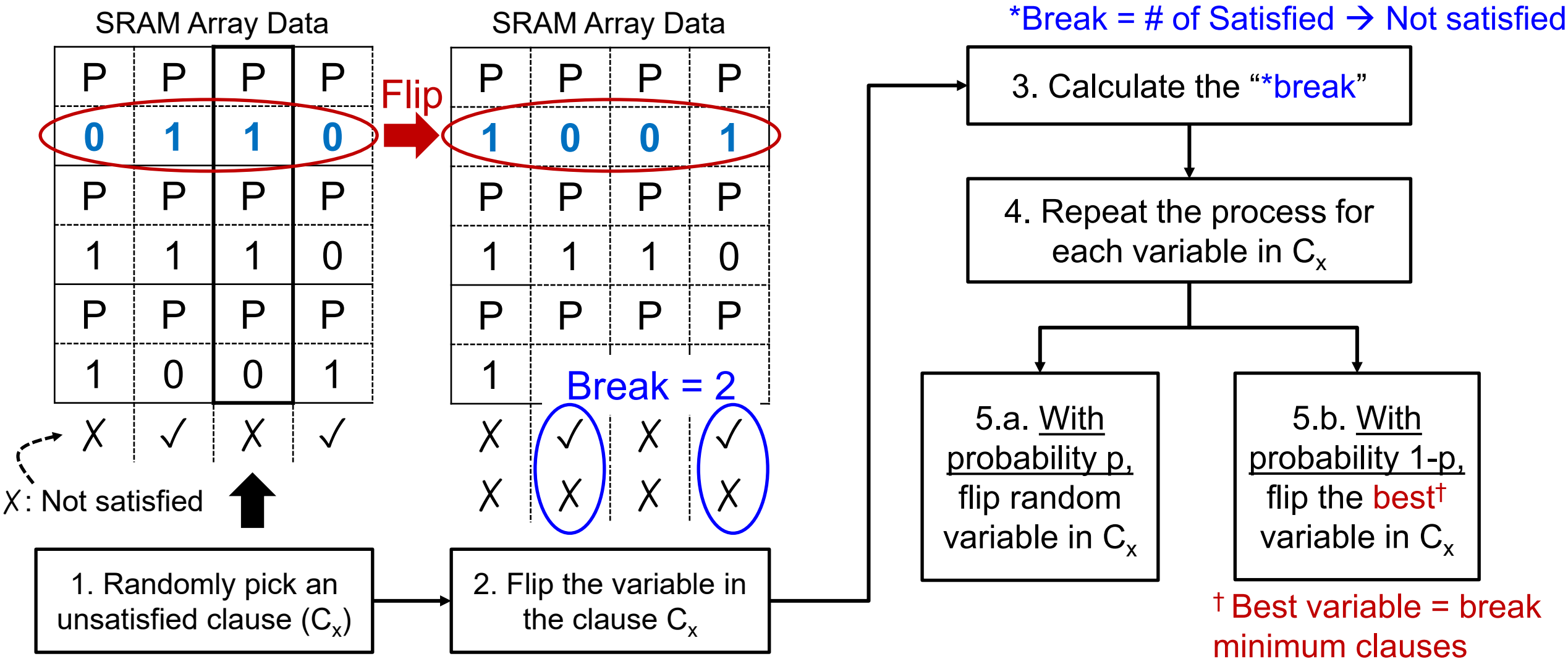


- Hardware friendly SAT algorithms
  - Variable flip
  - Solution search
- Write-after-Read operation
- Flip a variable = flip the entire row
- Repeat the whole process

Outline

- Motivation
  - Background
  - Prior boolean satisfiability problem (SAT) solvers
- Proposed Snap-SAT Architecture
  - Design highlights
  - SAT problem mapping example
  - Overall architecture
  - Circuit diagram
- **Silicon Prototype Measurements**
- Snap-SAT Summary

SAT Algorithm: WalkSAT Algorithm (WS)



Clause-to-Variable Ratio (CTV)

$$CTV = \frac{\#\ of\ Clauses}{\#\ of\ Variables}$$

A. Montanari, JSTAT, 2008

Clause-to-Variable Ratio (CTV)

CTV = 0.2

Number of clauses = 1

Number of variables = 5

$$F(x) = (\, x_0 \,\mathrm{OR}\, \overline{x_1} \,\mathrm{OR}\, x_5)$$

A. Montanari, JSTAT, 2008

Clause-to-Variable Ratio (CTV)

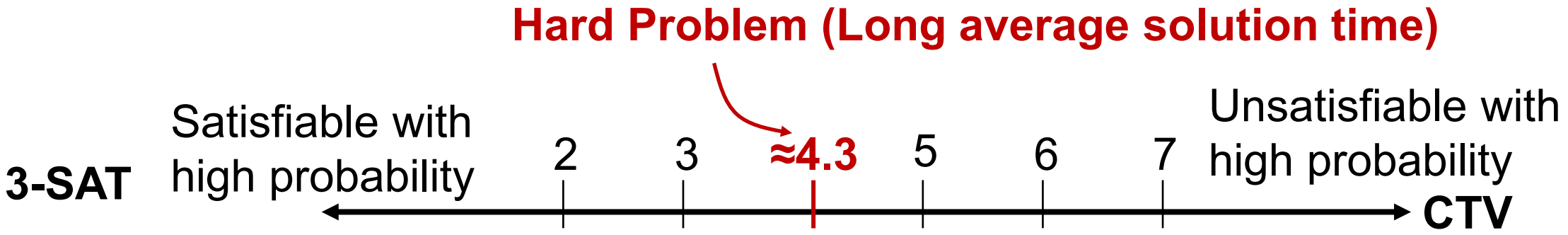
CTV = 200

Number of variables = 5

Number of clauses = 1000

$$F(x) = (\, x_0 \text{ OR } \overline{x_1} \text{ OR } x_5)$$
$$\text{AND } (x_0 \text{ OR } \overline{x_2} \text{ OR } \overline{x_4})$$
$$\text{AND } (x_2 \text{ OR } \overline{x_3} \text{ OR } x_5)$$
$$\text{AND } (\overline{x_2} \text{ OR } \overline{x_3} \text{ OR } \overline{x_4})$$
$$\text{AND } (\overline{x_1} \text{ OR } \overline{x_2} \text{ OR } x_3)$$
$$\text{AND } \ldots \textit{ to clause 1000}$$

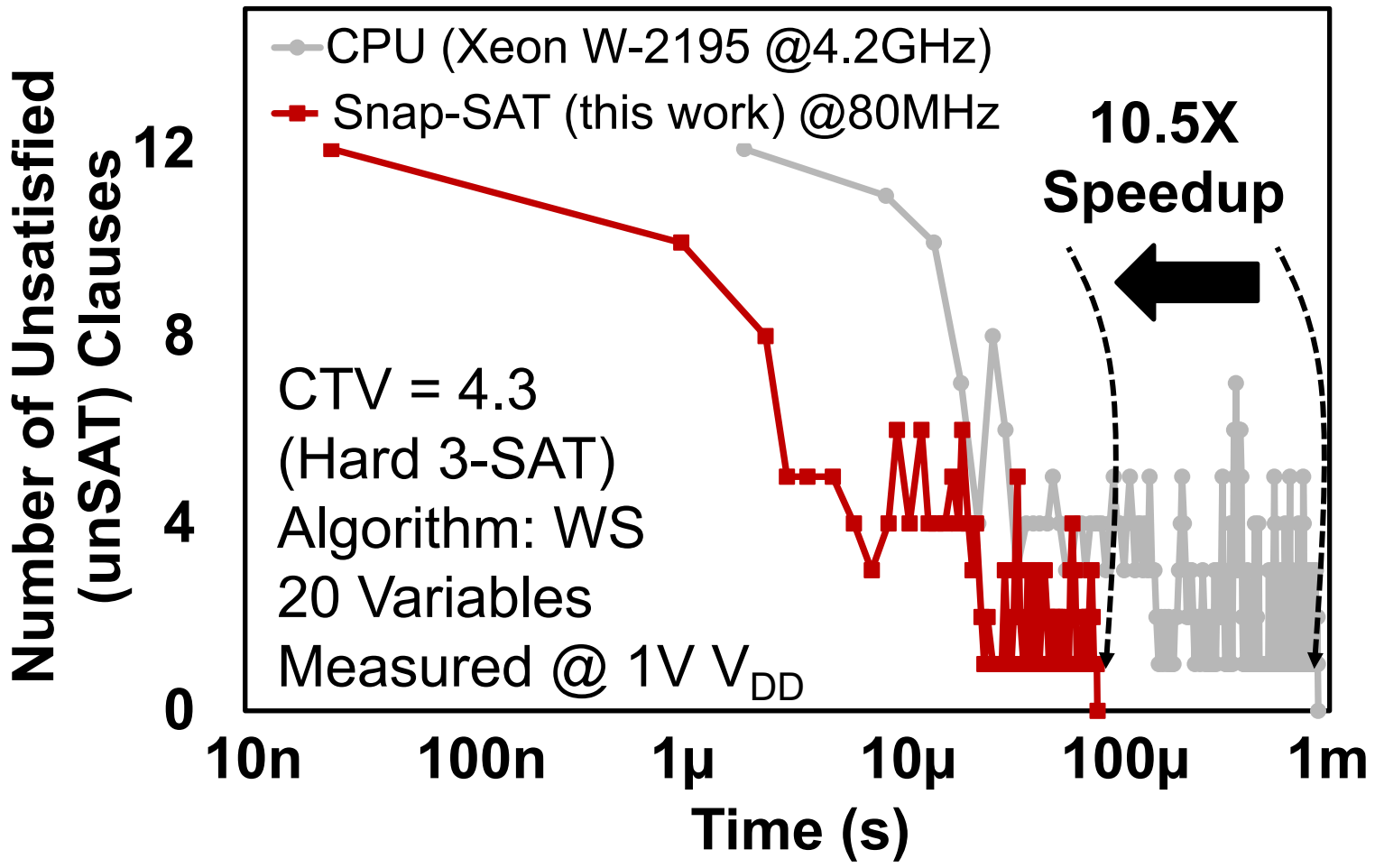
Clause-to-Variable Ratio (CTV)



	Hard Problem Regime		
	3-SAT	**4-SAT**	**5-SAT**
CTV	4.3	9.9	21.1

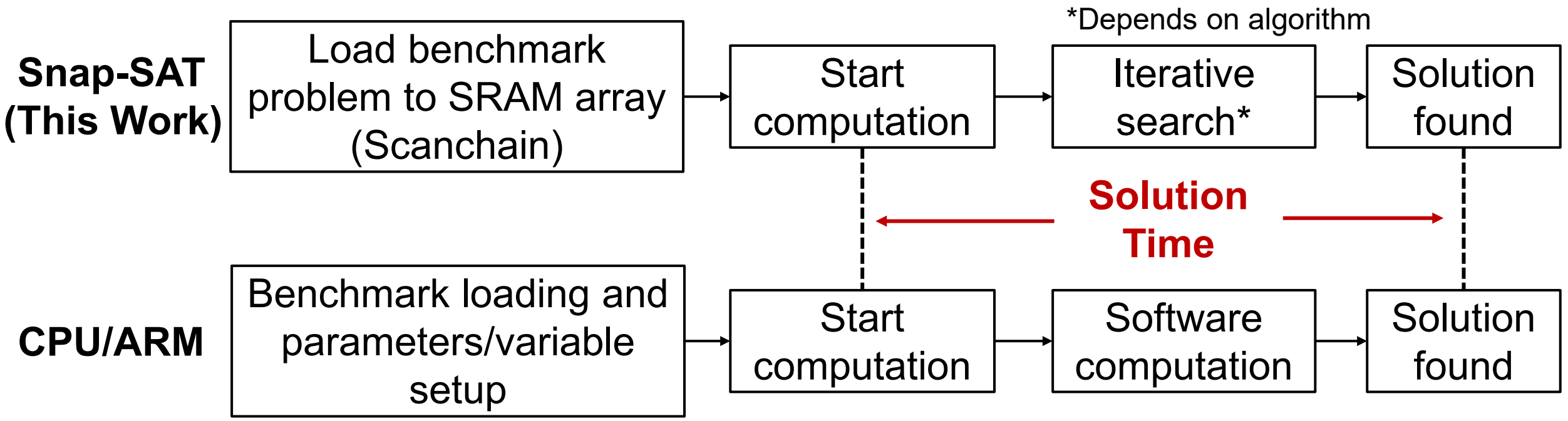
A. Montanari, JSTAT, 2008

Snap-SAT Computation Evolution

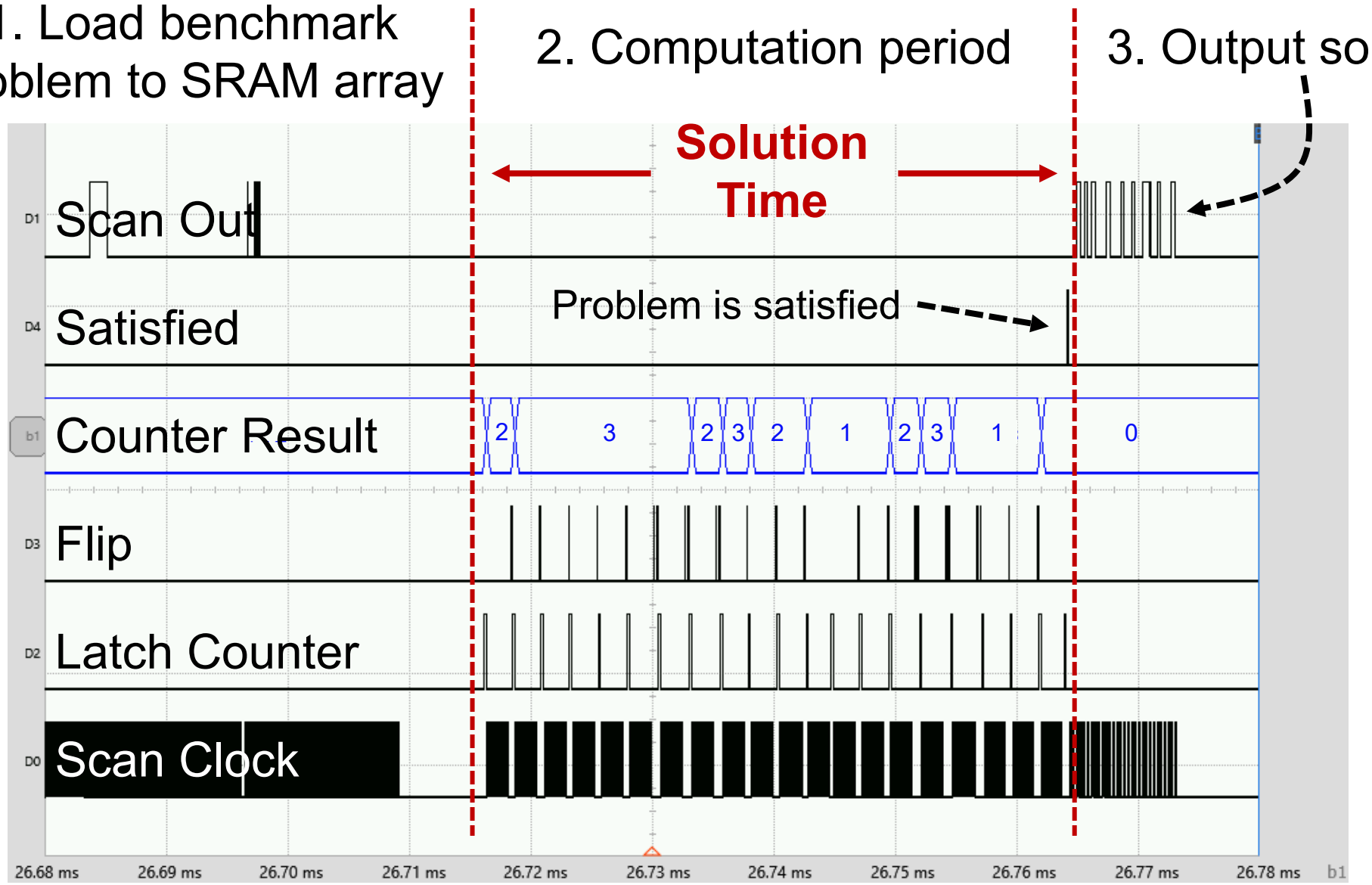


- 10.5X speedup compared to Xeon W-2195 CPU

Solution Time Definition

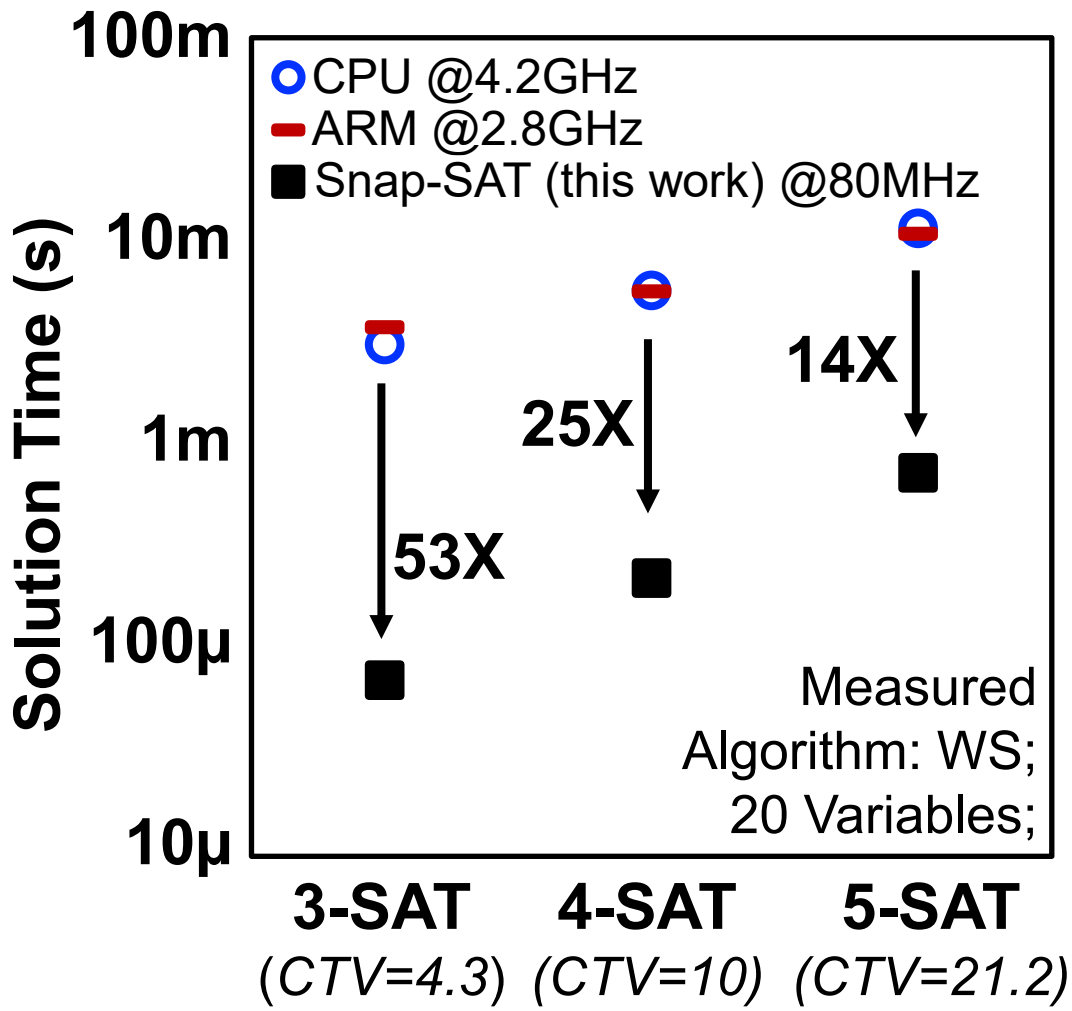


Snap-SAT Computation Flow Demonstration

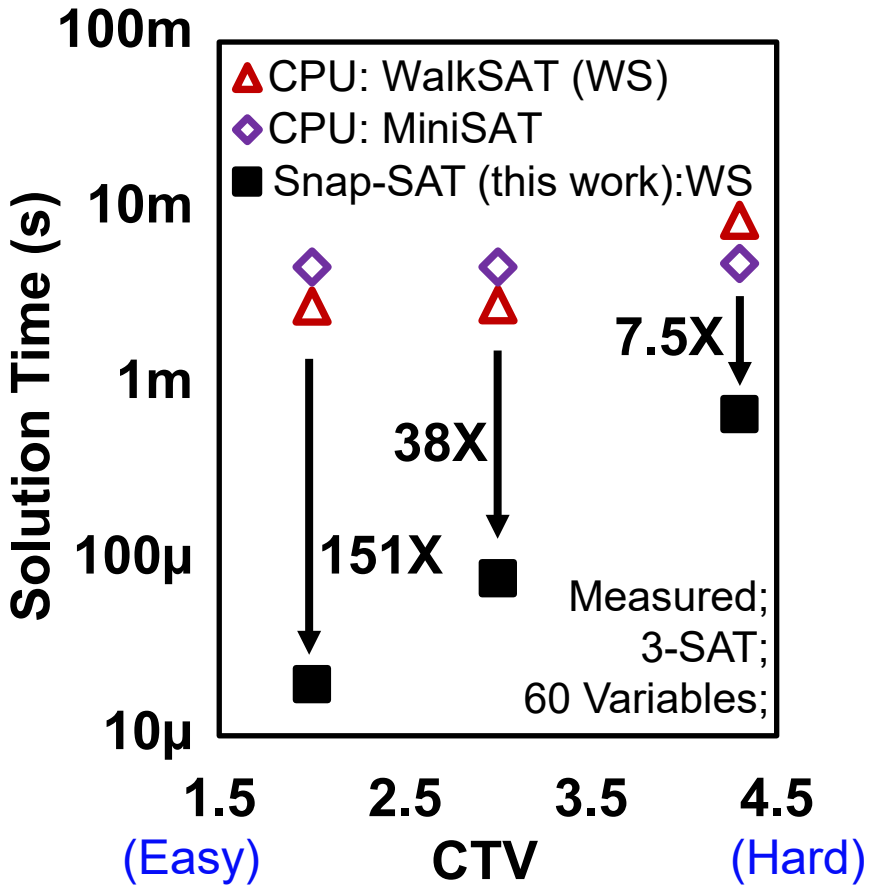


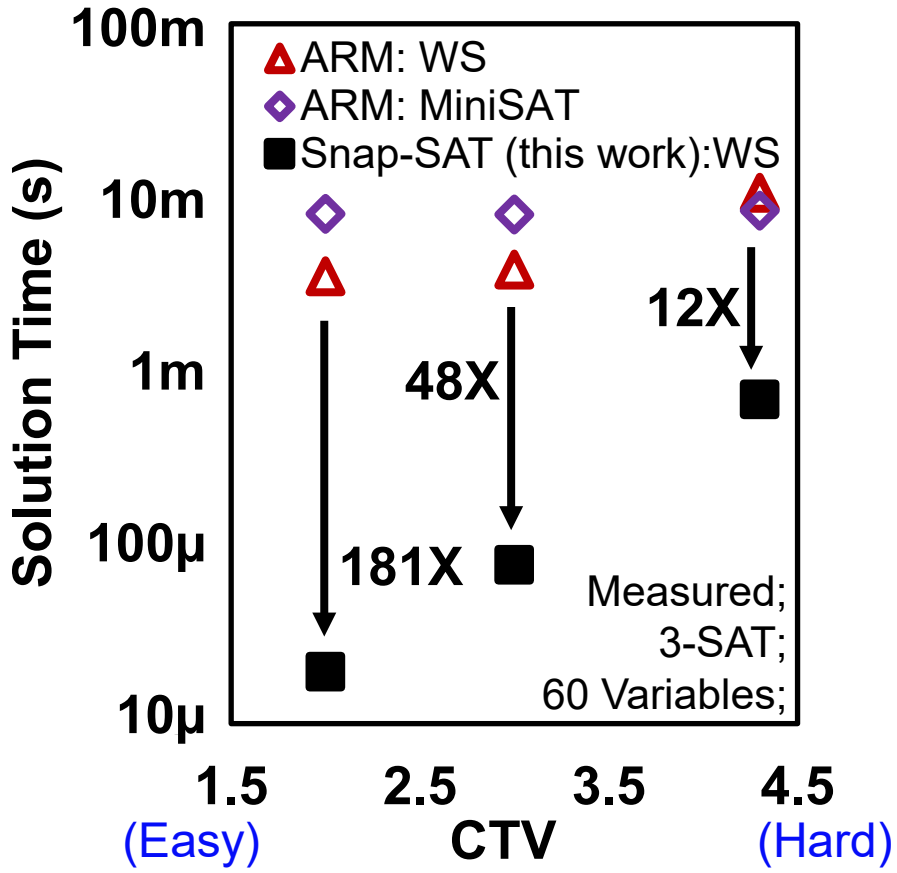
Solution Time Comparison

- 1000 SAT problems
- 1000 iterations/problem
- ISO-voltage comparisons: V_{DD}=1
- High performance processor:
  - CPU Xeon W-2195
- Energy-efficient processor:
  - ARM on LG-G710U device (Snapdragon 845)



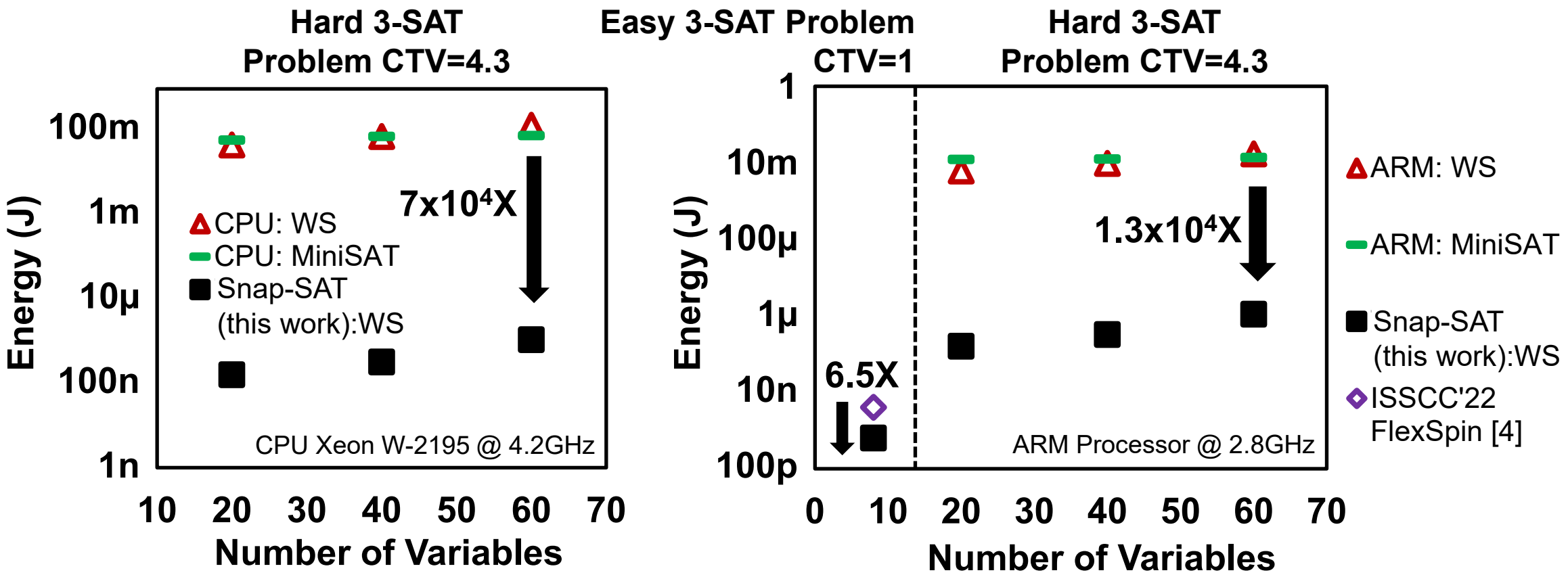
Solution Time Comparison





- MiniSAT: commonly used high-speed software algorithm
- MiniSAT: not hardware friendly / compute-in-memory friendly
- 7.5X (12X) speedup over CPU (ARM) at iso-V_{DD} while running 52X (35X) slower

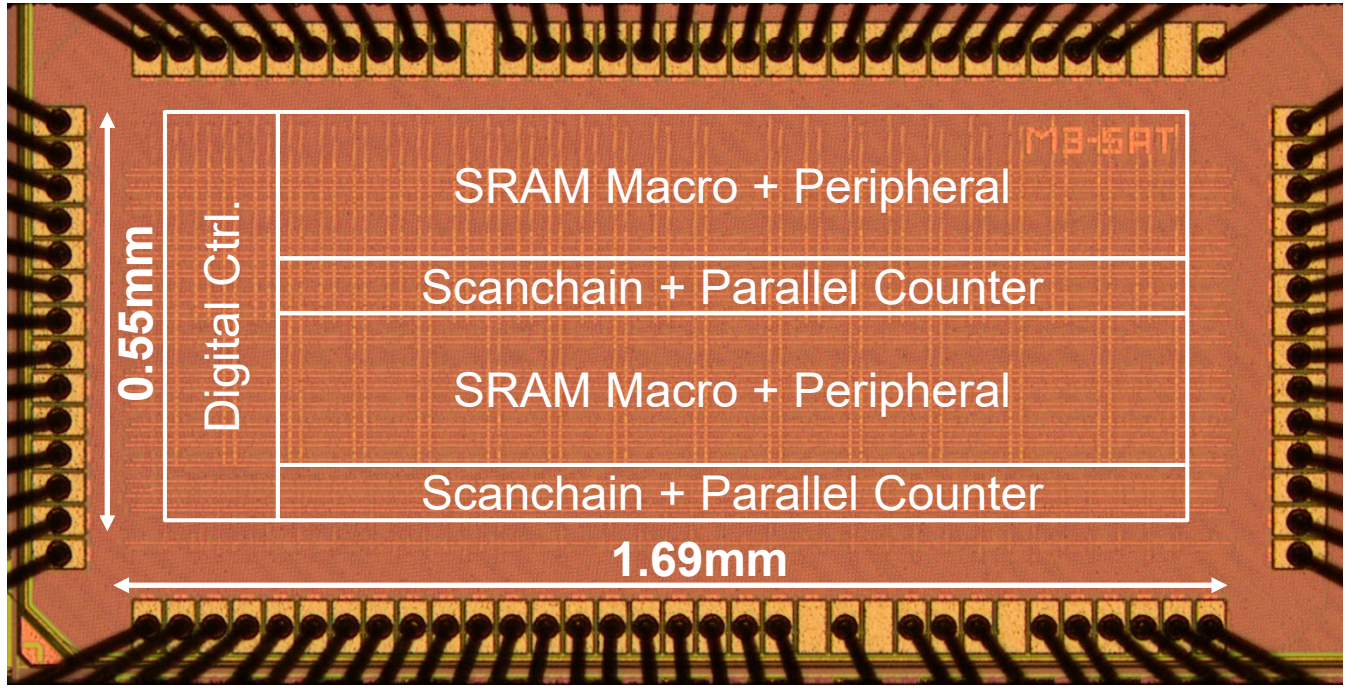
Energy Comparison



- Energy = Power * Solution Time
- CPU/ARM Power = Power (Running SAT) – Power (IDLE Condition)
- Single-core CPU operations for all algorithms

Chip Micrograph

- 65nm native CMOS process
- Memory capacity: 131Kb
  - Limited by test-chip size and fabrication cost
- Inside the chip:
  - Entire SAT computation flow
- Outside the chip:
  - Scan out computation details (clauses, counter results, etc.)
  - Other efficient algorithms



Comparison with Prior Works

	This work Snap-SAT	ISSCC'22 Flex-Spin [4]	CICC'22 [5]	Nature'15 [6]
CMOS Technology	65nm CMOS	65nm CMOS	65nm CMOS	180nm CMOS
Silicon Prototype	Yes	Yes	Yes	Yes
Compute Type	Digital	Digital	Analog	Digital & Analog
Computation Mechanism	Column-wise In-memory OR Operation	Dedicated Processing Element	Continuous-Time Dynamic	Oscillator Pulse
In-Memory Computation	Yes	Yes	No	No
Frequent SRAM Access	Not Required	Not Required	Required	Required

Comparison with Prior Works

		This work Snap-SAT	ISSCC'22 Flex-Spin [4]	CICC'22 [5]	Nature'15 [6]
Hardness of Problem Tested		Hard (CTV=4.3)	Easy	Hard	Hard
Maximum Number of Variables		128 ([4]Scalable w/ row size)	8	50	50 (Test) 200 ([2])
Maximum Number of Clauses		1024 ([4]Scalable w/ column size)	8	212	218 (Test) 1260 ([2])
Accuracy		100%	-	93.5%	-
Average Solution Time (3-SAT)	CTV=4.3 (Hard)	0.71ms (Variables=60)	-	0.9ms (Variables=10)	7.5ms (Variables=50)
	CTV=1 (Easy)	0.41µs (Variables=8)	5µs (Variables=8)	0.15µs (Variables=10)	-
[1]Energy (nJ) (V_{DD}=1V)	CTV=4.3 (Hard)	1098	-	[3]21600	-
	CTV=1 (Easy)	0.63	4.1	[3]3.6	-

[1]Energy = Measured Average Solution Time * Measured Average Computation Power; [2]Reported without source of data (simulation or measurement)

[3]Reported peak power = 27mW (1.1V)\; Energy number is scaled to 1V. [4]Row/Column size is limited by the prototype cost

Silicon Prototype Summary

Design Details	
Technology	65nm CMOS
Supply Voltage	0.7V~1.2V
Memory Capacity	131Kb
Chip Area	0.93mm^2
Frequency	80MHz
***k*-SAT Problem Parameters**	
k parameter (*k*-SAT)	2~128
Maximum Variable Size (n)	128 (Scalable w/ row size)
Maximum Clause Size (M)	1024 (Scalable w/ column size)
Measurement Results Algorithm: WalkSAT; *k*-SAT = 3-SAT; 60 Variables; CTV = 4.3 (Hard Problem)	
[1]Solution Time	713 µs
[2]Energy @V_{DD}=1V	1098 nJ
[3]Solvability	72%
[4]Accuracy	100% (All-digital design)

[1]End to end process time from start to solution found including scanchain in/out duration
[2]Energy = Measured Average Solution Time * Measured Average Computation Power
[3]Number of SAT problem that can be solved in a given time
[4]Correctness of the solved SAT problem

Outline

- **Motivation**
  - Background
  - Prior boolean **sat**isfiability problem (SAT) solvers
- **Proposed Snap-SAT Architecture**
  - Design highlights
  - SAT problem mapping example
  - Overall architecture
  - Circuit diagram
- **Silicon Prototype Measurements**
- **Snap-SAT Summary**

Summary

- Accurate all-digital in-memory computation
- Single-cycle massive parallel computations
- Scalable to large and hard SAT problems
- 7.5X (12X) solution time speed up over CPU (ARM)
- $7*10^4$X ($1.3*10^4$) energy reduction over CPU (ARM)
- Demonstrated on different hard *k*-SAT problems with different variable sizes, clause sizes, and CTV ratios
- A promising, fast, reliable, reconfigurable, and scalable compute-in-memory design for solving SAT problems

Acknowledgments

- **TSMC university shuttle support**
- **NSF CAREER award**
- **Micron foundation faculty grant**
- **AMD chair endowment**
- **UT Austin Circuit Research Lab (https://sites.utexas.edu/CRL/)**
  - Advisor: Professor Jaydeep Kulkarni
- **Fangming Ning** 🥰
  - Demo front end software development
  - Demo session 2 (demonstrated last night)

Thank you for your attention!