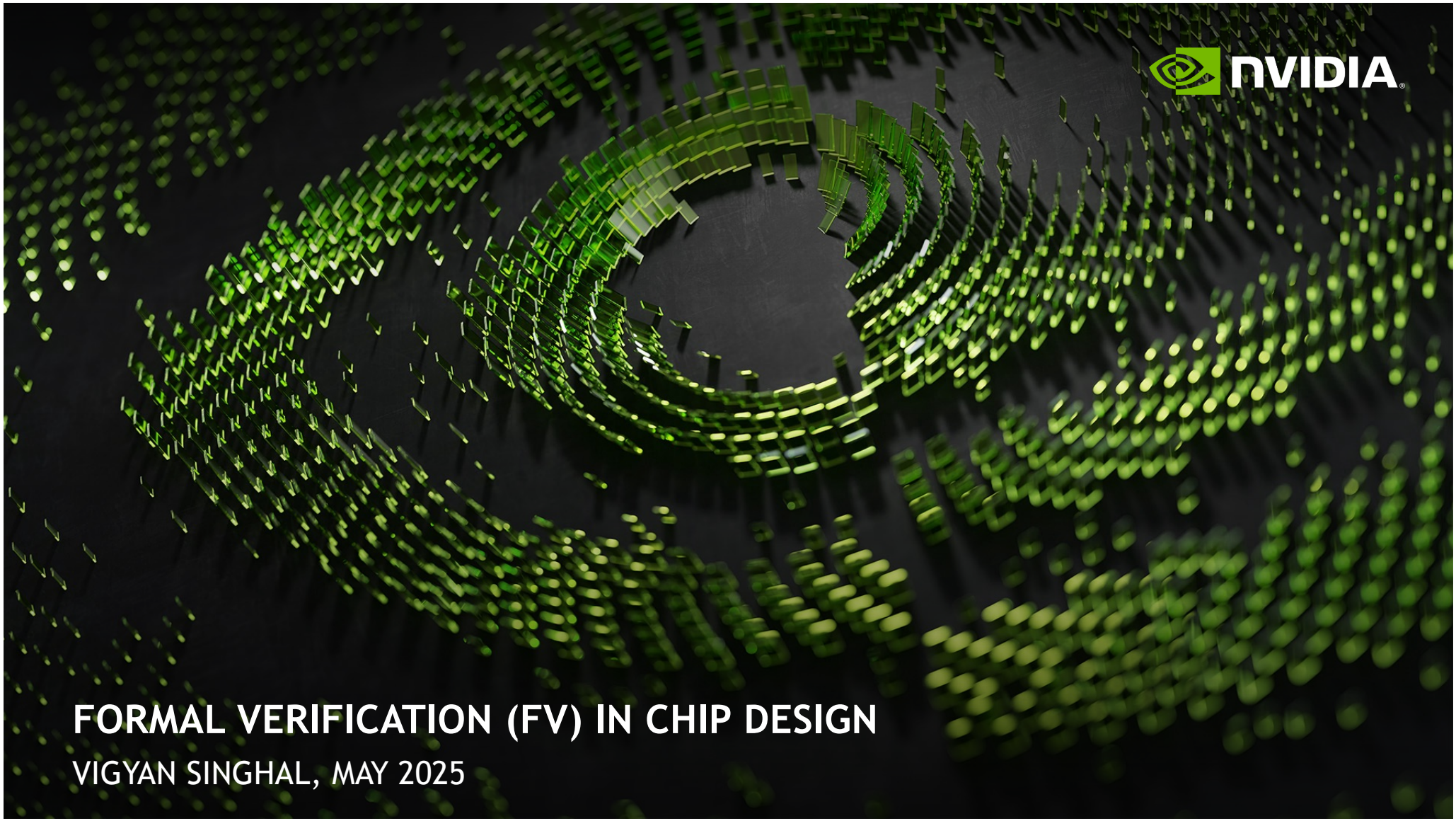


FORMAL VERIFICATION (FV) IN CHIP DESIGN

VIGYAN SINGHAL, MAY 2025

SPEAKER INTRODUCTION

- BTech, CS, IIT Kanpur, 1989
- Almost quit after 3 years trying to get an MS, UC Berkeley, 1992
- MS, CS, 1994; PhD, CS, 1996
  - Design replacements for sequential circuits
- Cadence Berkeley Labs, 1995-99
- Jasper, founder and 1st CEO, 1999-2005
  - Struggled to raise VC funding for 3 years; so, boot-strapped
  - Leading FV model checking tool in chip companies
- Elastix, founder and CEO, 2006-07
  - Failed to productize asynchronous clocking; became Marvell's European design center (Barcelona)
- Oski, founder and 1st CEO, 2008-2021
  - Acquired by NVIDIA; Gurugram (India) and Budapest design centers

BOB BRAYTON (1933-2025)

- BS, EE, Iowa State, 1956
- PhD, Math, MIT, 1961
- IBM, Math and Sciences department, 1961-87
  - On the asymptotic behavior of the number of trials necessary to complete a set with random selection
- UC Berkeley, 1987-
- Father of logic synthesis
  - Synopsys founders were Bob's students
- Started studying FV in 1991
- Software tools
  - IBM tools, ESPRESSO, SIS, VIS, ABC
- Phil Kaufman award, 2007
- Oski BMC Deep Bounds award, 2017

MAJOR FV BREAKTHROUGHS (1971-1999)

- 1977: Temporal logic of programs, Amir Pnueli. Turing Award, 1996.

- 1982: Automatic verification of finite-state concurrent systems,
Ed Clarke, Allen Emerson, JP Queille, Joseph Sifakis. Turing Award, 2007.

- 1986: An automata-theoretic approach to automatic program verification,
Moshe Vardi, Pierre Wolper. Gödel Prize, 2000.

- 1986: Graph algorithms for Boolean function manipulation (BDDs), Randy Bryant.
Phil Kaufman Award, 2007.

- 1987: Symbolic model checking, Ken McMillan. LICS Test of Time Award, 2010.

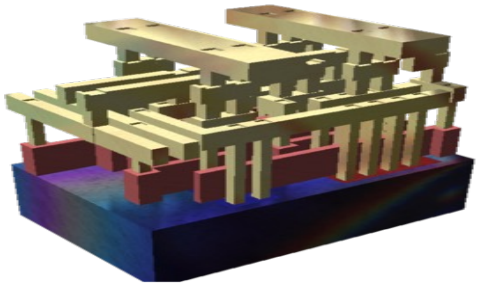
- 1999: Symbolic model checking without BDDs,
Armin Biere, Alessandro Cimatti, Ed Clarke, Yunshan Zhu. CAV Award, 2018.

CHIP DESIGN FLOW

```
always @ (posedge clk)
  if (sel) q <= d1;
  else q <= d2;
assign p = d1 & d2 | d3;
```

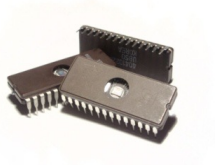
Hand-written
RTL design

EDA tools →



IC layout database
(GDS II)

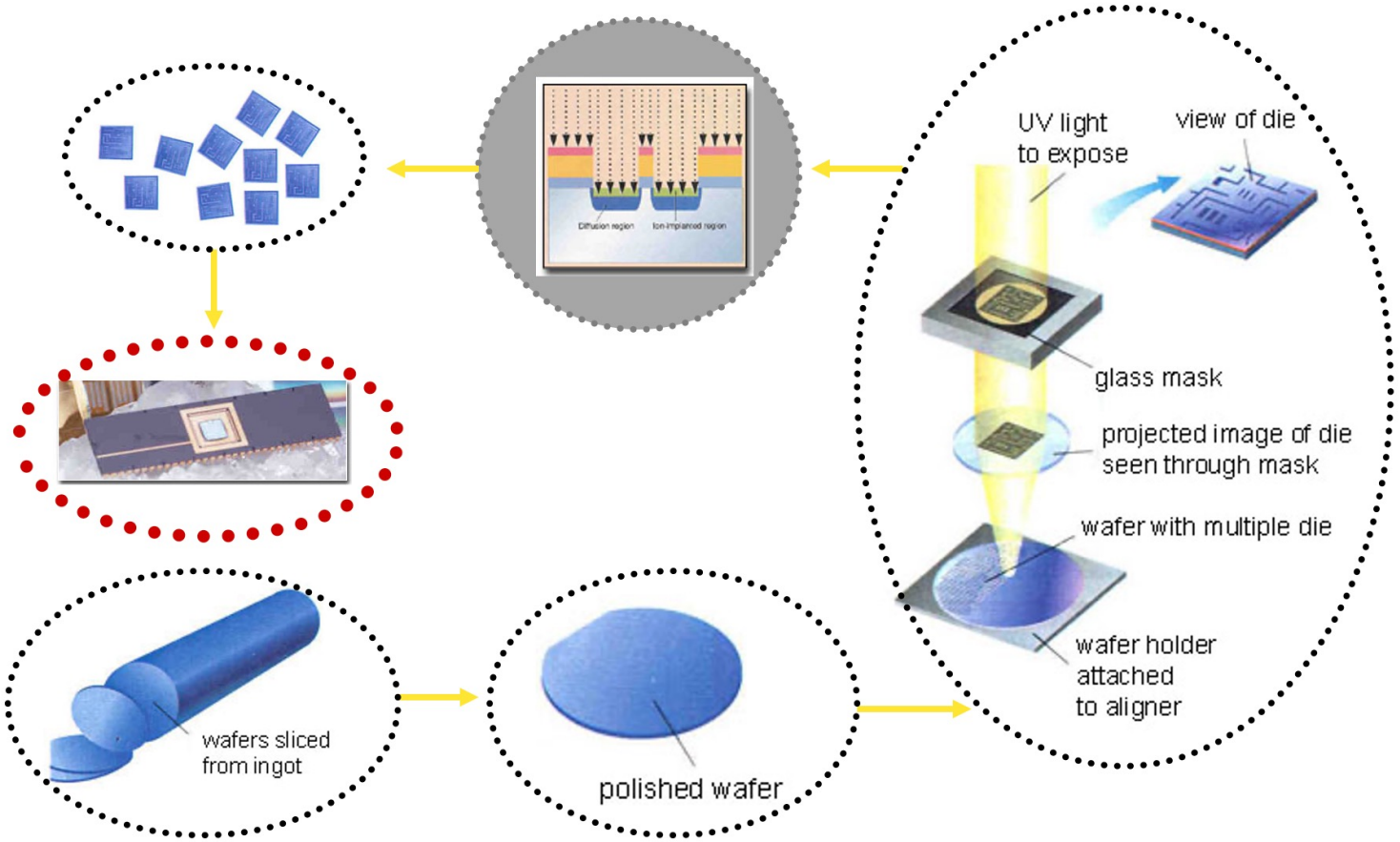
manufacturing →



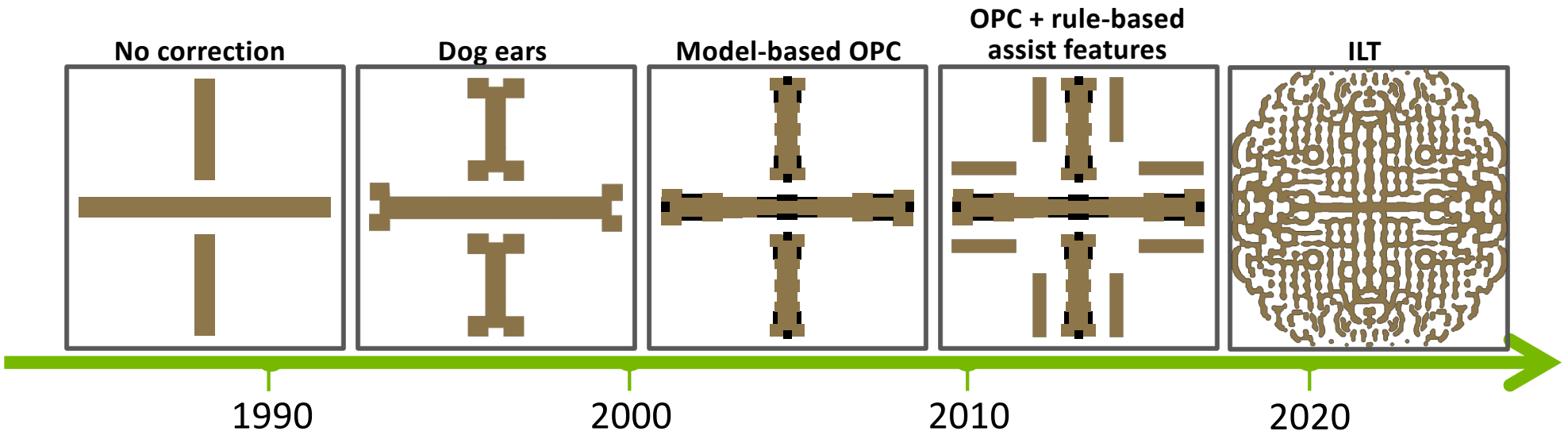
Electronic chips

- About 20-50 different EDA tools in any design flow
- Tools help improve area (cost), delay (frequency), power and correctness of designs
- Important areas: synthesis, logic verification, timing verification, placement, routing, signal integrity, …

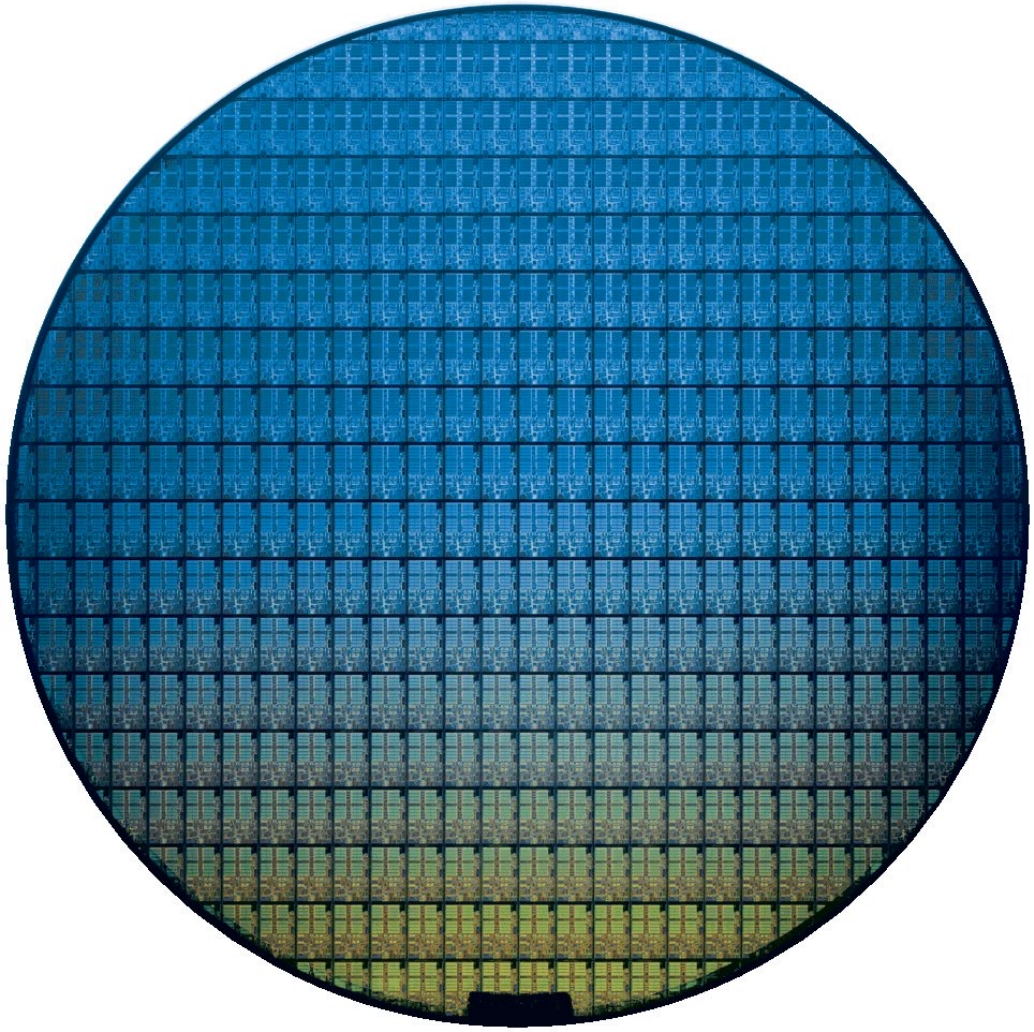
MANUFACTURING A CHIP



INCREASING NEED FOR MASK CORRECTION

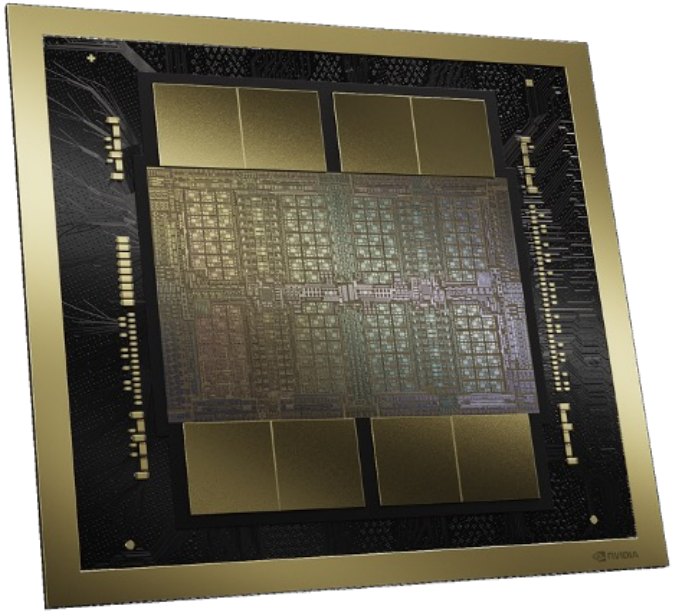


- From simple decorations to complex “distortions”
- Intuition finally breaks down



NVIDIA

TODAY'S COMPLEX CHIPS



- NVIDIA Blackwell GPU (2024)
- 208B transistors
- TSMC 4nm process
- 1600 mm^2
- 8 TB/s memory bandwidth
- 20 PetaFLOPS FP4 AI
- FV is critical to silicon success


NVIDIA
SOME HISTORY

THE INTEL PENTIUM FDIV BUG

- Introduced by Intel in March 1993
  - First superscalar x86
- In June 1994, Thomas Nicely discovers a bug in division
  - 4,195,835/3,145,727 = 1.333820449136241002 (in reality)
  - 4,195,835/3,145,727 = 1.333739068902037589 (per Pentium)
- In December 1995, Intel recalls all Pentium chips
  - Incurs a charge of $475 Million

- Hardware formal verification got a lot of attention ($) from:
  - Intel, and some other chip companies
  - Venture Capitalists (VCs), funding EDA startups

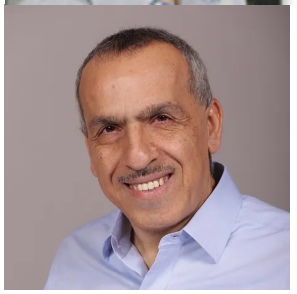
COMMERCIAL FV TOOLS IN 1999

- Large EDA tools
  - IBM: RuleBase (1993)
  - Cadence: FormalCheck (before IFV)
    - From Bob Kurshan's COSPAN (Bell Labs)
    - Later, IFV
  - Synopsys: Magellan
    - Later, VC Formal
- Internal tools in chip companies
  - IBM, Intel, Motorola, …
- Startups (VC investments)
  - 0-in (acquired by Siemens)
  - Atrenta Periscope
  - Axiom Design Automation
  - Averant
  - Chrysalis
  - Real Intent
  - Verplex BlackTie
  - Verysys (Ed Clarke was an advisor)

Many of these tools were based on SMV (CMU) or VIS (Berkeley)
Many used BDD packages (UC Boulder) or SAT solvers (Chalmers)

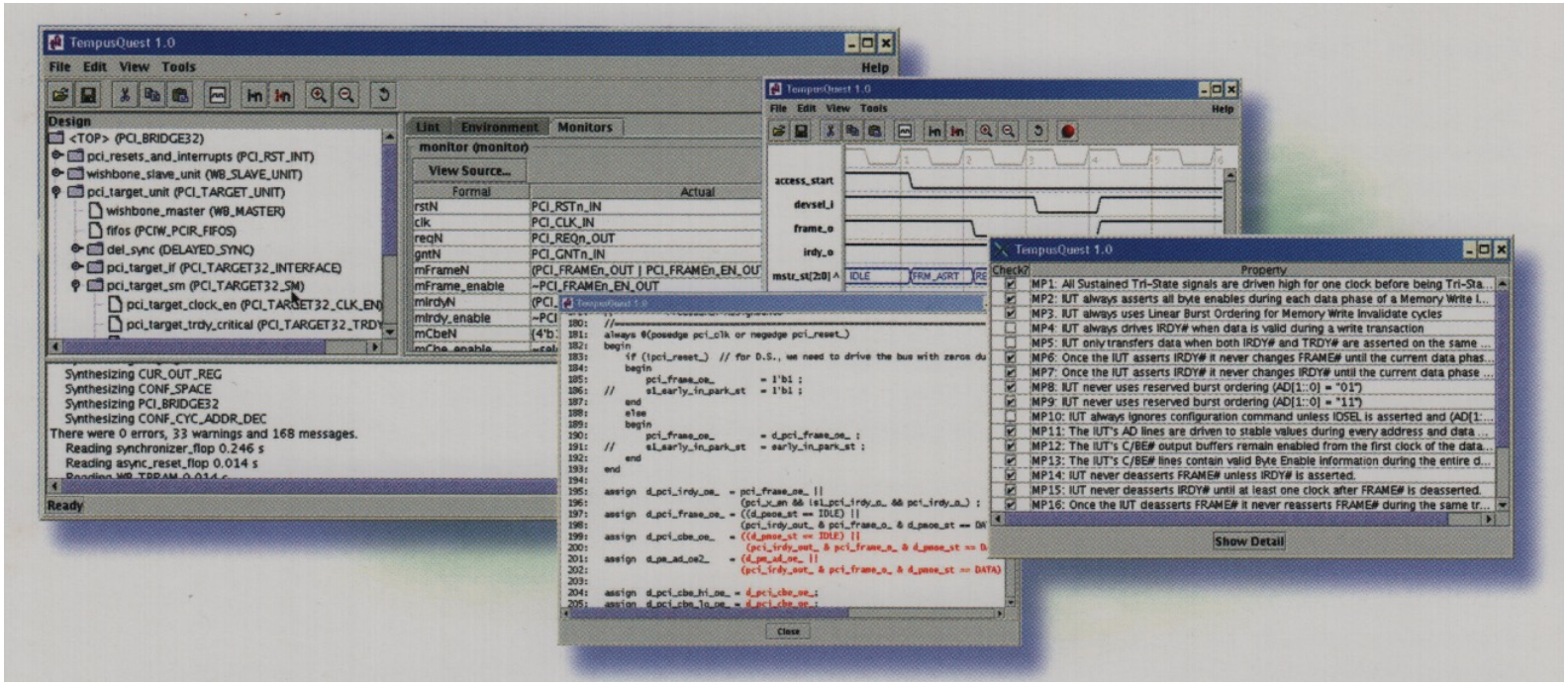
JASPER HISTORY

From 1999



- Founded in 1999 (Joe Higgins and Vigyan)
- Failed to raise VC $ (more than 10 competitors)
- Decided to differentiate on usability and GUI
  - In the early years, made sure no one evaluated on engine performance! ☺
  - Bundled tool with a user for many years (sold bugs, not tool)
- Proof-of-concept at NVIDIA
- NVIDIA investment of $XXX in 2002 (for 6 future licenses of tool!)
  - Saved the company
- Raised VC money ($7.1 Million) in 2003 and hired sales CEO (Kathryn Kranen)
- Hired Niklas Eén (MiniSat) to take a break from his PhD, and rewrite engines in C!
- Acquired Safelogic in Sweden in 2004 (developers, engines and PSL compiler)
- New engineering team in Israel, headed by Ziyad Hanna, starting in 2007
- A few more VC $ investments…
- Cadence bought Jasper for $170 Million in 2014

JASPER ORIGINAL GUI (CA. 2002)



Sole developer: Joe Higgins

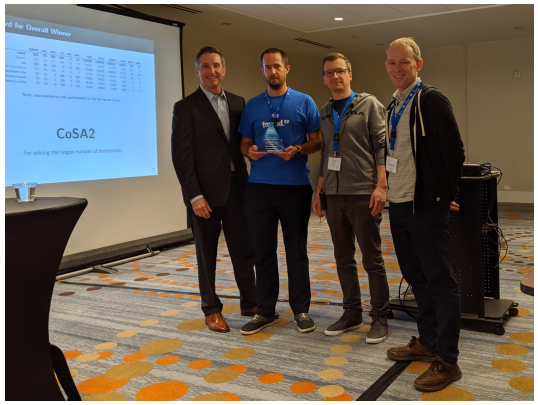
SOME MORE FORMAL BREAKTHROUGHS SINCE 1999

Surely, I am missing many other important papers (esp. since 2011)

- 1999: Checking safety properties using induction and a SAT solver, Sheeran, Singh, Stålmarck.
- 2000: Counterexample-guided abstraction refinement, Clarke, Grumberg, Jha, Lu, Veith.
- 2001: Interpolation and SAT-based model checking, McMillan.
- 2011: SAT-based model checking without unrolling, Bradley.
- 2024: Memory-efficient multi-GPU accelerated explicit state space exploration with GPUexplore 3.0, Wijs, Osama.

ENGINE COMPETITIONS

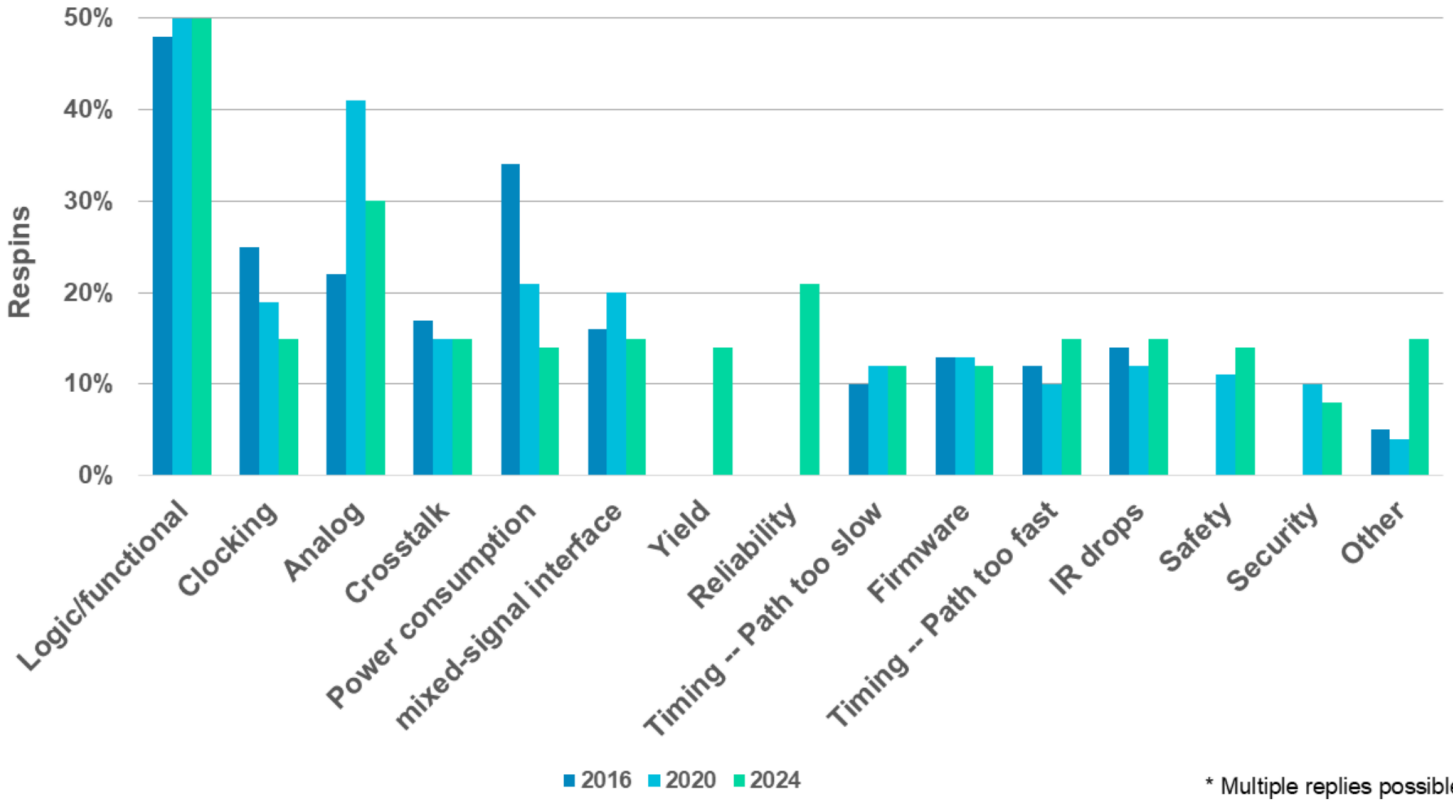
- Hardware Model Checking Competition (HWMCC)
  - Every 1 or 2 years from 2007
  - Bounded Proof track sponsored by Oski 2012-2019
  - Winners are typically from universities
  - Engine performance impressive (vs commercial tools)



- SAT Competition
  - Annually from 2002

- SMT-COMP
  - Annually from 2005

POST-SILICON FLAWS ACROSS INDUSTRY*

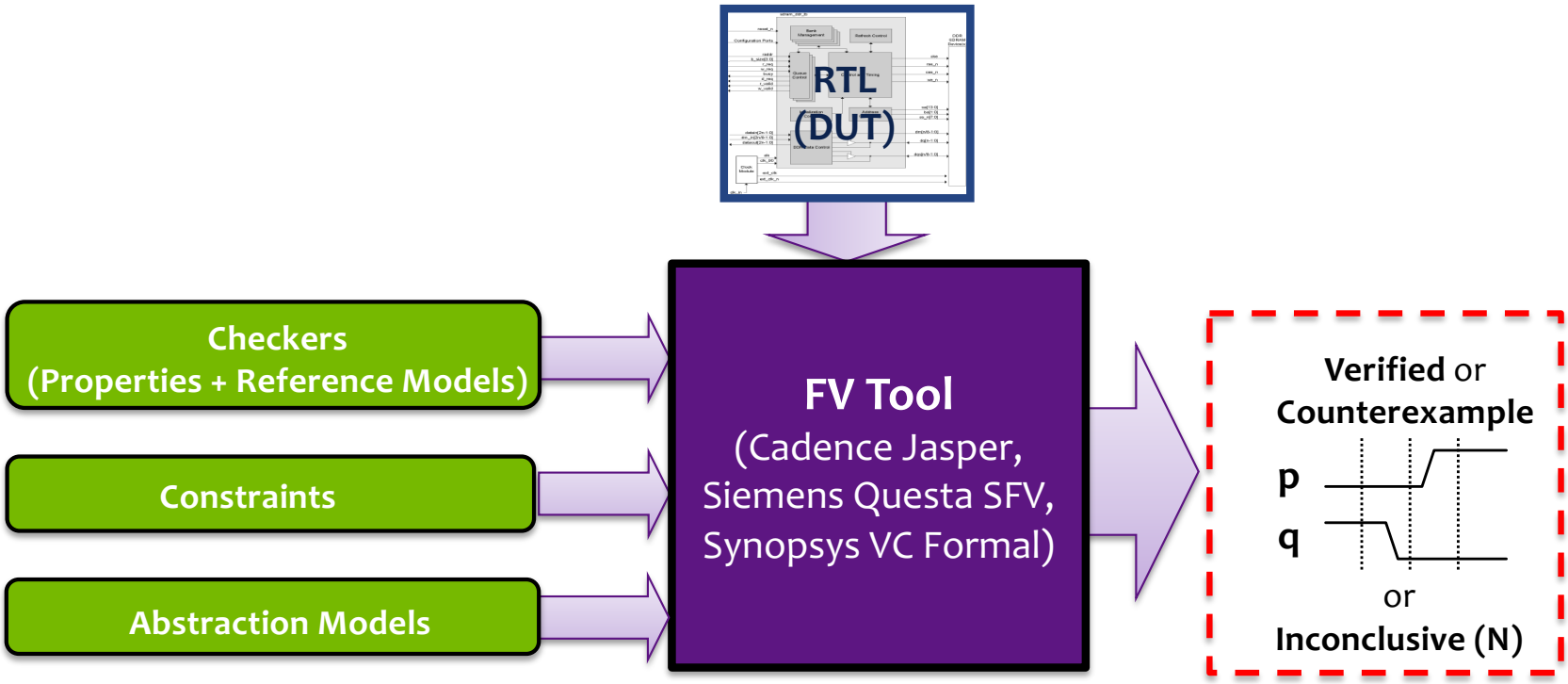


*2024 Wilson Research Group report

NVIDIA


NVIDIA
FV UNDER-THE-HOOD

FV TOOL USE MODEL

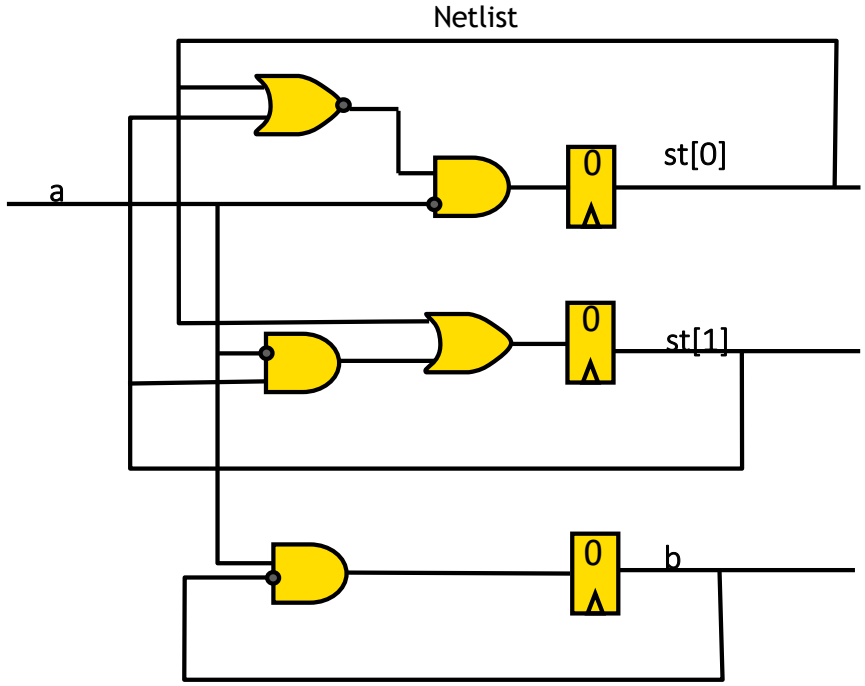


RTL TO NETLISTS

RTL (Verilog)

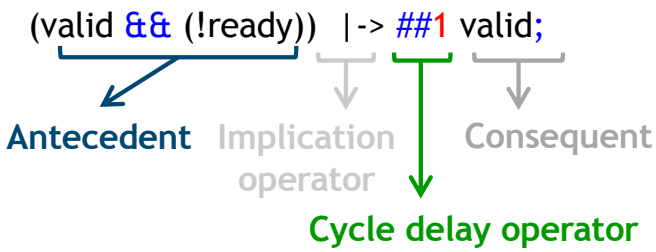
```
module test (input a, output b);

reg b;
reg [1:0] st;

always @(posedge clk or negedge rst)
  if (~rst) st <= 2'b00;
  else
  case (st) // full_case
    2'b00: if (~a) st <= 2'b01;
    2'b01: st <= 2'b10;
    2'b10: if (a) st <= 2'b00
  endcase

always @(posedge clk or negedge rst)
  if (~rst) b <= 1'b0;
  else if (~a | b) b <= 1'b0;
  else b <= 1'b1;

endmodule
```

synthesis

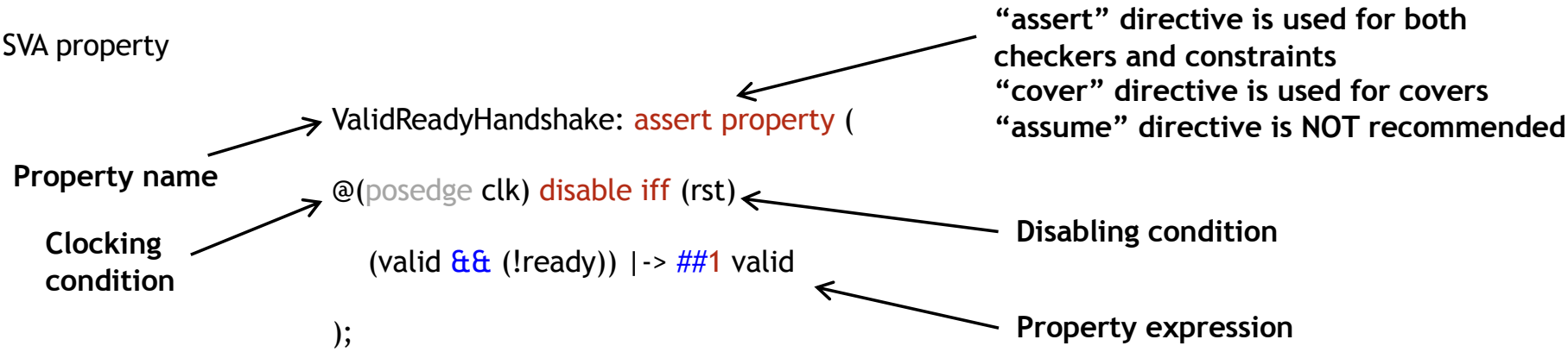


ENGLISH PROPERTY TO SYSTEMVERILOG (SVA)

- Valid-ready handshake protocol: If *valid* is asserted, it should remain asserted until *ready* is received
- SVA expression

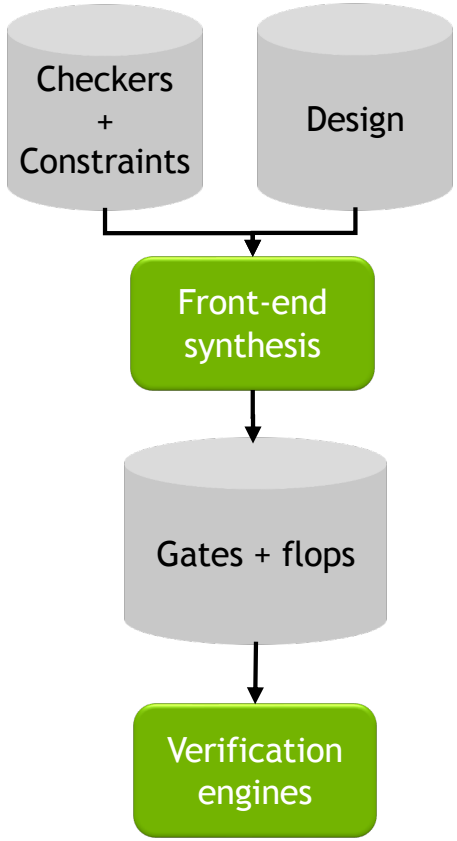


- SVA property

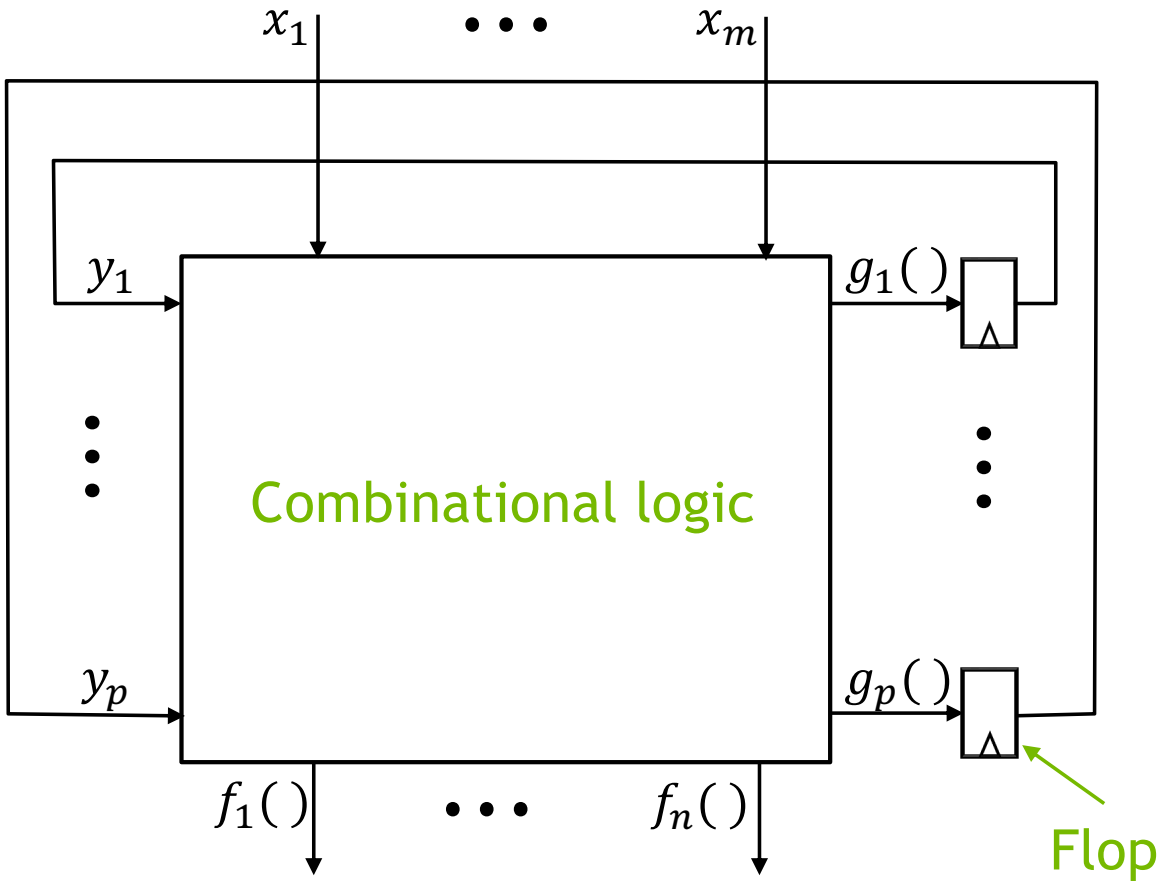


UNDER-THE-HOOD…

- Synchronous front-end
  - Design synthesized to gates+flops
    - But not optimized
  - Checkers, constraints also synthesized
    - Synthesizing general SVA is non-trivial

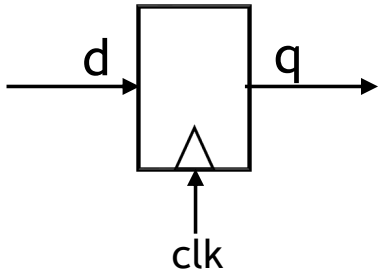


NETLISTS



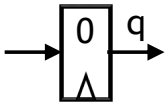
- Flops (p)
- Combinational logic
  - m primary inputs: x_i
  - p state inputs: y_k
  - p next-state outputs
    $g_k(x_1, \cdots, x_m, y_1, \cdots, y_p)$
  - n primary outputs
    $f_j(x_1, \cdots, x_m, y_1, \cdots, y_p)$
  - **Acyclic** network of gates, e.g. AND, OR, NOT
- Inputs and functions are Boolean

FLOP

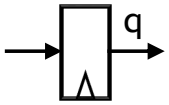


$q_{@(t+1)} = d_{@t}$

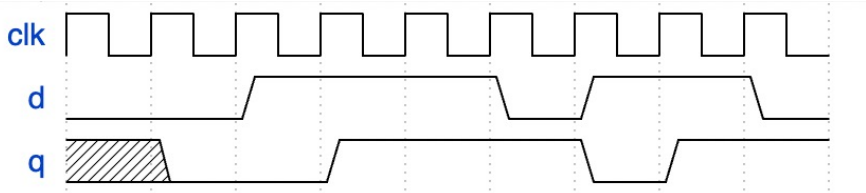
- Implements notion of time
- Two types of flops
  - Reset flops
    - $q_{@0}$ is 0



  - Non-reset flops
    - $q_{@0}$ is either 0 or 1 (denoted by "X")
    - better area, delay



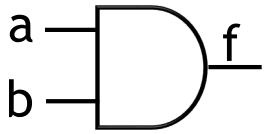
time	@0	@1	@2	@3	@4	@5	@6	@7	@8
d	0	0	1	1	1	0	1	1	0
q	X	0	0	1	1	1	0	1	1



COMBINATIONAL LOGIC GATES

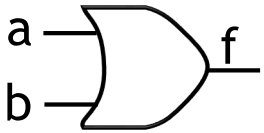
- Determines primary output and next-state functions
- Each function is an acyclic network of gates, e.g. AND, OR, NOT

AND



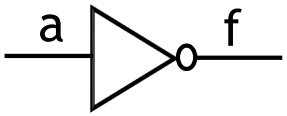
$f_{@t} = a_{@t} \wedge b_{@t}$

OR



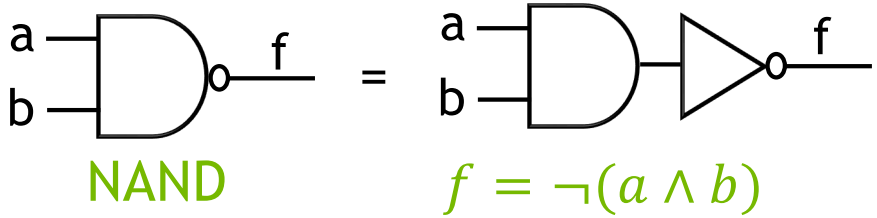
$f = a \vee b$

NOT



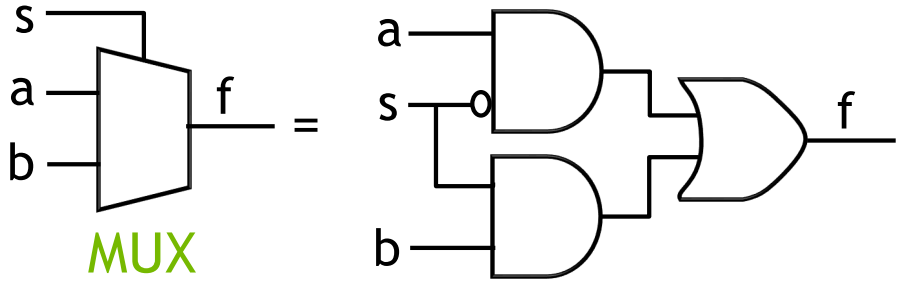
$f = \neg a$

SECONDARY GATES (EXAMPLES)



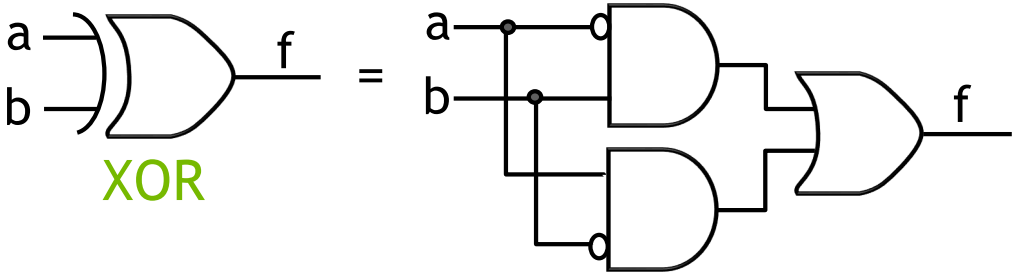
NAND

$f = \neg(a \wedge b)$



MUX

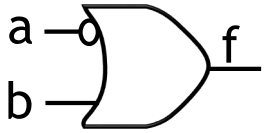
$f = (\neg s \wedge a) \vee (s \wedge b)$



XOR

$f = a \oplus b = (\neg a \wedge b) \vee (a \wedge \neg b)$

IMPLIES



$f = (a \rightarrow b) = \neg a \vee b$

HOMEWORK PROBLEMS*

1. Show that any combinational Boolean function $f : \{0,1\}^n \to \{0,1\}$ can be computed using the following set of logic gates: 2-bit AND, 2-bit OR, and NOT.

2. Show that any combinational Boolean function $f : \{0,1\}^n \to \{0,1\}$ can be computed using the following single logic gate: 2-bit NAND.

3. Show that there are infinitely many Boolean functions $f : \{0,1\}^n \to \{0,1\}$ cannot be computed using the following set of logic gates: 2-bit XOR, and NOT.

*stolen from Ryan O'Donnell's course

WASON PROBLEM (FROM SHANKAR'S TALK)

- Given four cards laid out on a table as: $\mathbf{D}$, $\mathbf{3}$, $\mathbf{F}$, $\mathbf{7}$, where each card has a letter (L_i) on one side and a number (N_i) on the other
- Which cards should you flip over to determine if every card with a $\mathbf{D}$ on one side has a 7 on the other side?

$$f_i = (L_i = \mathbf{D}) \rightarrow (N_i = \mathbf{7})$$

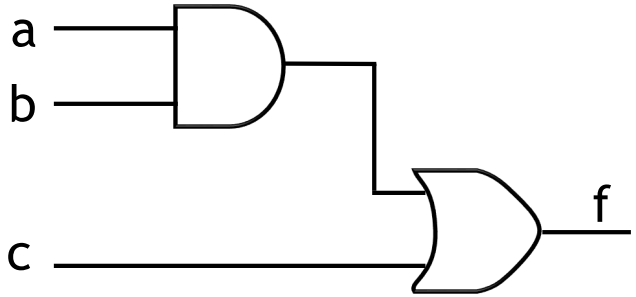
$$f_i = (L_i \neq \mathbf{D}) \vee (N_i = \mathbf{7})$$

BOOLEAN ALGEBRA

- Monotonic laws
  - $x \wedge 1 = x$
  - $x \wedge 0 = 0$
  - $x \wedge x = x$
  - $x \wedge y = y \wedge x$
  - $x \wedge (y \wedge z) = (x \wedge y) \wedge z$
  - $x \wedge (x \vee y) = x$
  - $x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z)$
  - …
- Non-monotonic laws
  - $x \wedge \neg x = 0$
  - $\neg(\neg x) = 0$
  - $\neg x \wedge \neg y = \neg(x \vee y)$
    (De Morgan's law)
  - …

COMBINATIONAL CIRCUIT-SAT

- Is there a Boolean assignment to inputs that evaluates a net to 1?



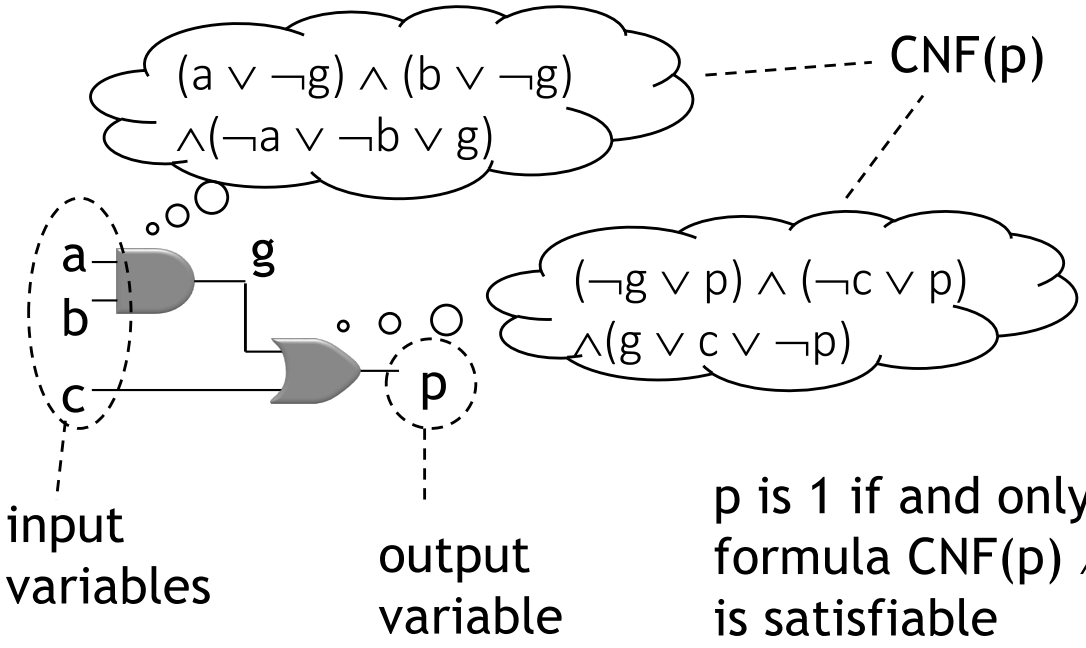
$$f = (a \wedge b) \vee c$$

SATISFIABILITY CHECKING (SAT)

- Boolean variables: $x_1, x_2, x_3, \ldots$
- Literals: $x_1, \neg x_1, x_2, \neg x_2, \ldots$
- Clauses: e.g. , $\neg x_1, (\neg x_1 \vee x_2), (\neg x_1 \vee x_2 \vee x_3), \ldots$
- Formulas in CNF: e.g.
  - $\alpha = (x_1 \vee \neg x_2) \wedge (x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee \neg x_3)$
  - $\beta = (x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (\neg x_2 \vee \neg x_1) \wedge x_1$
- Given a truth assignment to variables, we can compute the value of a formula, e.g.
  - Given $\{x_1, x_2, x_3\} = \{1, 0, 1\}$, the value of α is 0
- A formula is satisfiable (SAT) if a truth assignment exists that it to 1, else it is unsatisfiable (UNSAT), e.g.
  - β is UNSAT; α is SAT: Given $\{x_1, x_2, x_3\} = \{1, 1, 1\}$, the value of α is 1
- Satisfiability checking is NP-Complete (Cook, 1971)

TSEITIN'S TRANSFORMATION

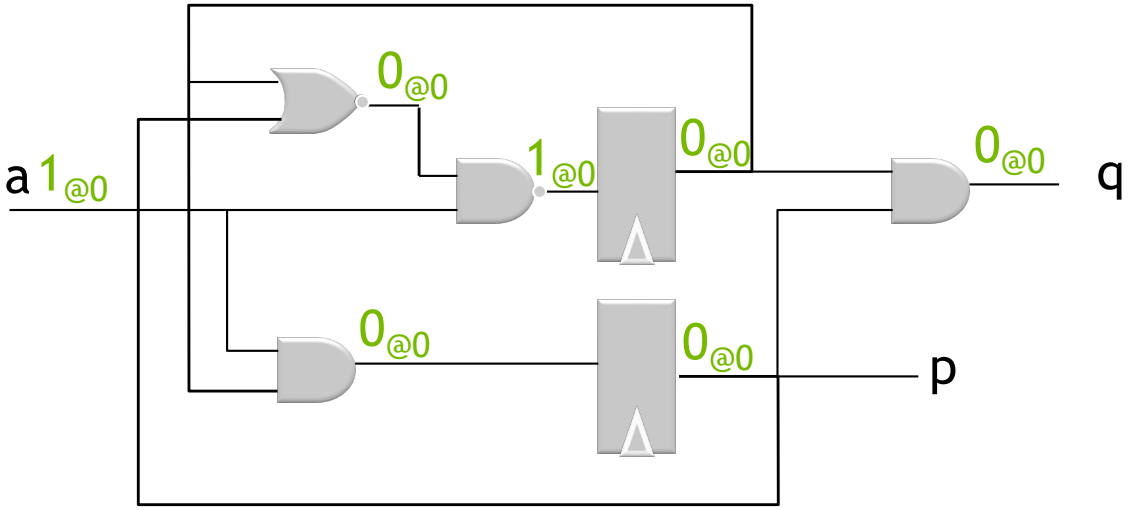
Can circuit output "p" be 1?



Circuit Satisfiability is NP-Complete

SEQUENTIAL CIRCUIT-SAT

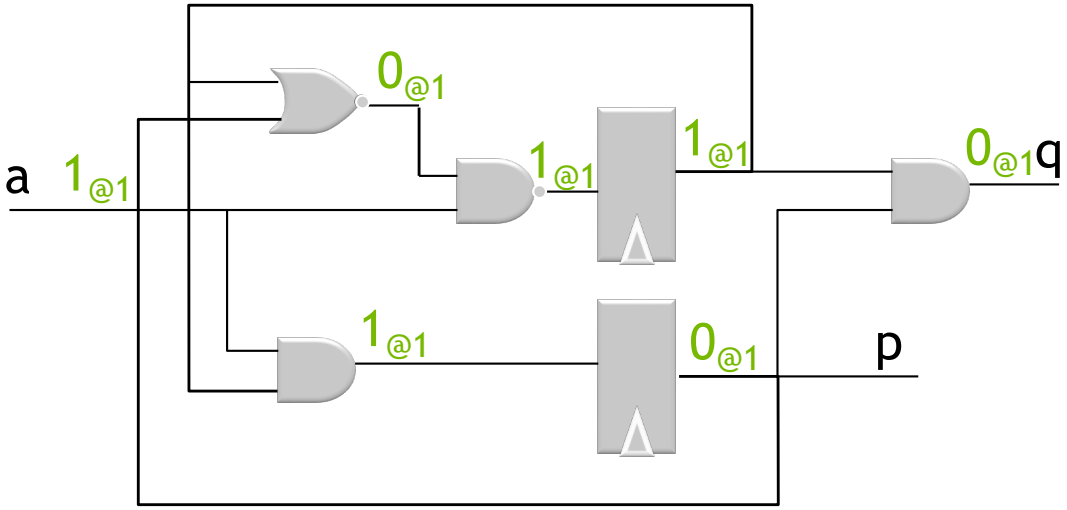
- Is there a sequence of input assignments such that p is 1 at any finite time?



Combinational gates + flops
Each flop has a reset value of 0

SEQUENTIAL CIRCUIT-SAT

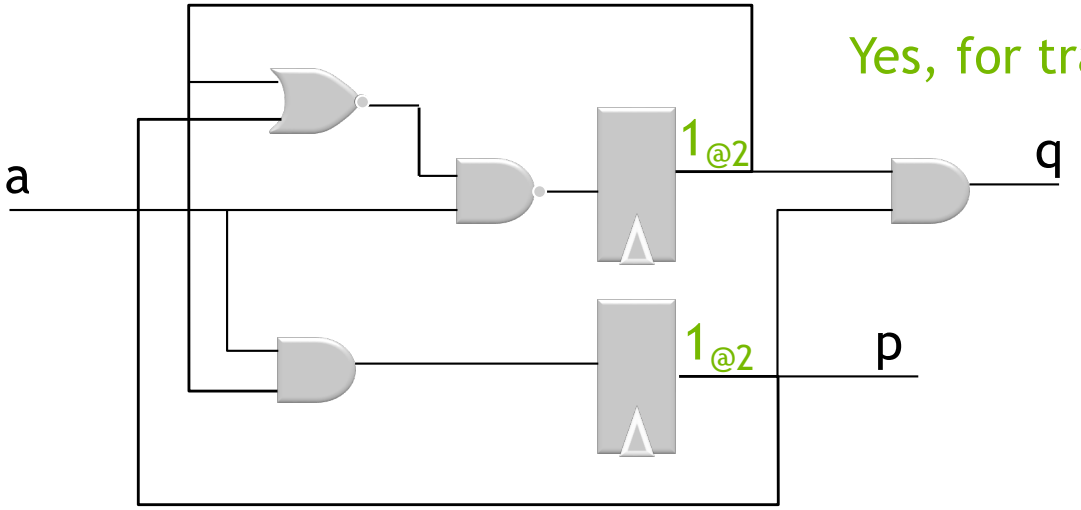
- Is there a sequence of input assignments such that p is 1 at any finite time?



Combinational gates + flops
Each flop has a reset value of 0

SEQUENTIAL CIRCUIT-SAT

- Is there a sequence of input assignments such that p is 1 at any finite time?



Yes, for trace a = $1_{@0}$, $1_{@1}$

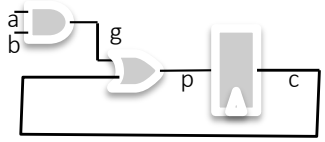
Combinational gates + flops

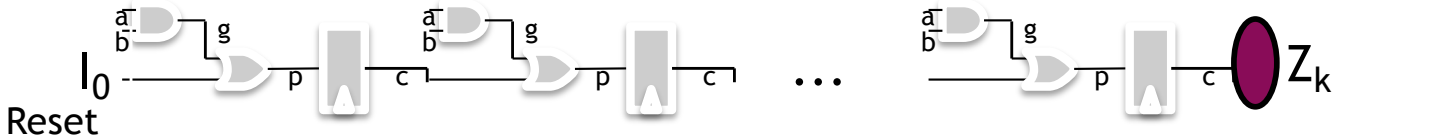
Complexity: PSPACE-complete (Aziz/Singhal/Brayton, 1993)

BOUNDED MODEL CHECKING (BIERE 99)

- Unfold the design k times:

$$U_k = C_0 \wedge C_1 \wedge \ldots \wedge C_{k-1}$$





Reset states

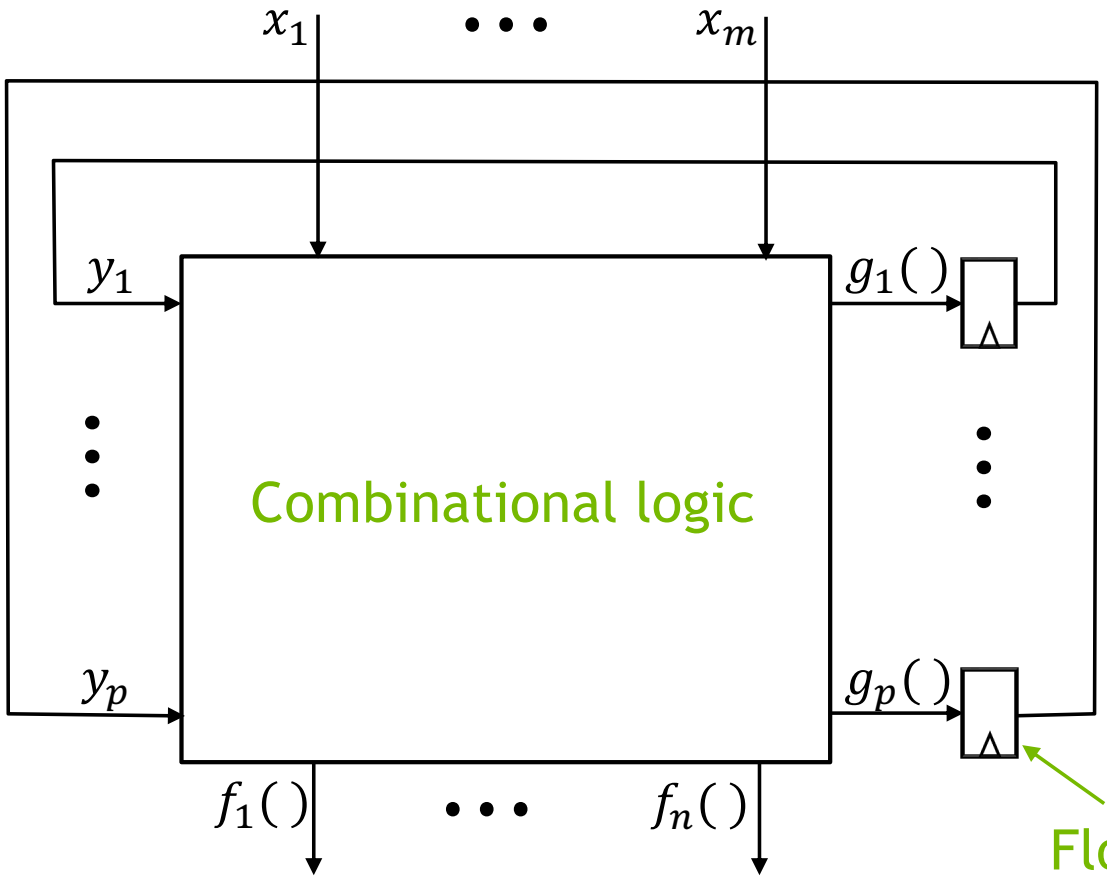
Bad states

- Use SAT solver to check satisfiability of

$$I_0 \wedge U_k \wedge Z_k$$

- A satisfying assignment is a counterexample of k steps
  - If we encounter a bad state at depth n < k, then we will encounter that bad state also at depth k and at any depth greater than k
- Non-satisfiable means it is impossible for this design to fail in k steps or less

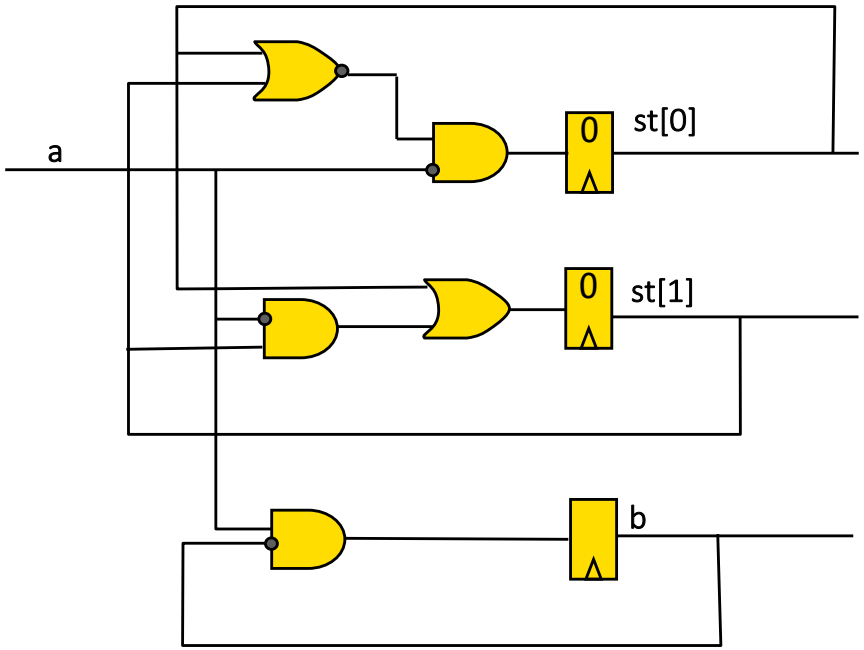
STATE SPACE



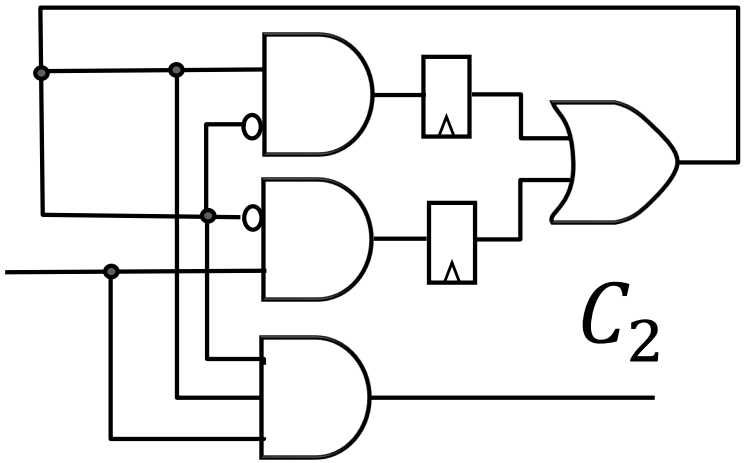
- 2^m input values per cycle
- STG with 2^p states
- q ($\leq p$) non-reset flops
- 2^q initial (reset) states

- Even for a small design, state space is huge

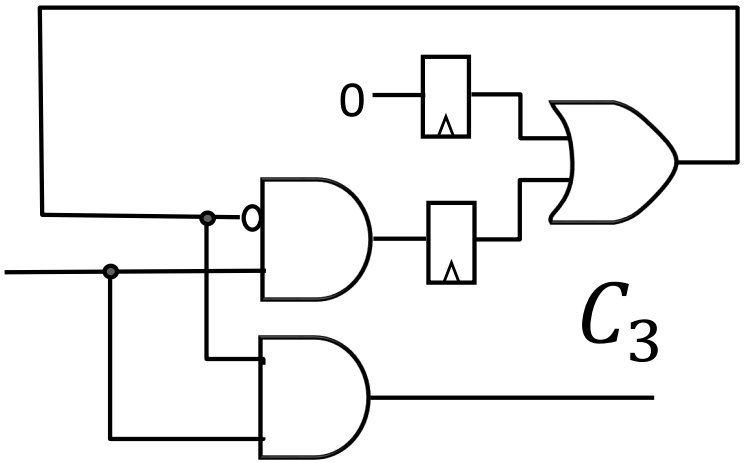
STATE-TRANSITION GRAPHS

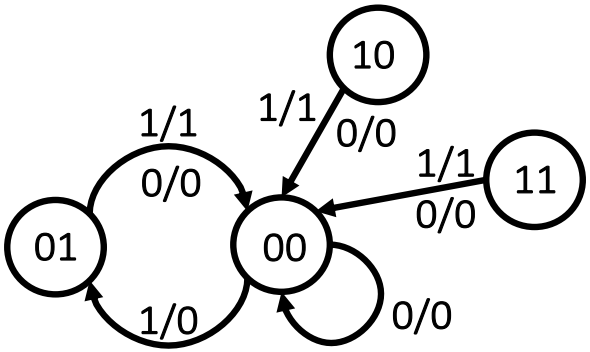


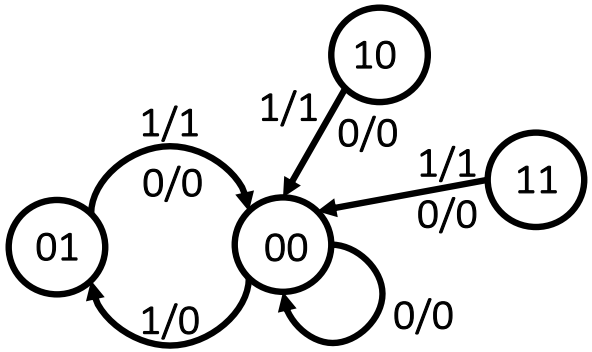
SEQUENTIAL EQUIVALENCE (MULTIPLE RESET STATES)



?

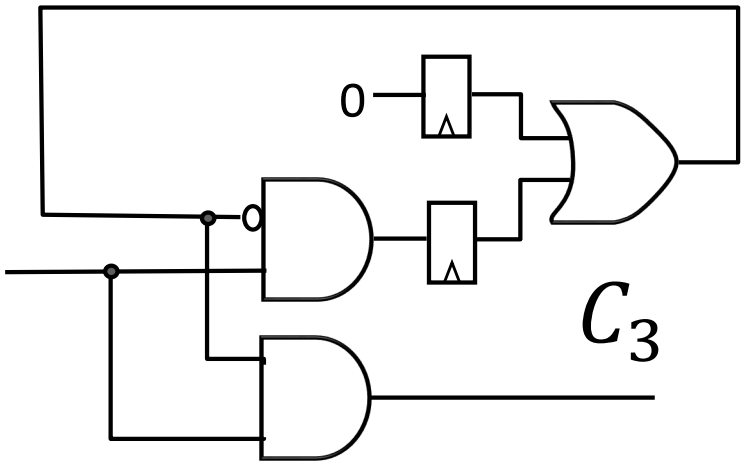




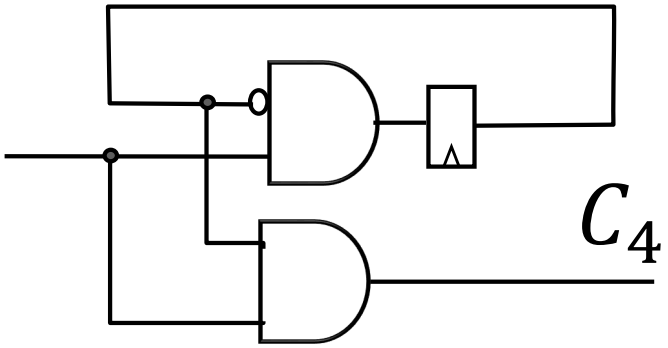


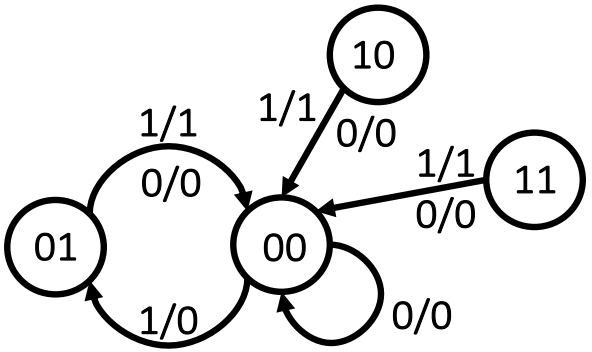
NVIDIA

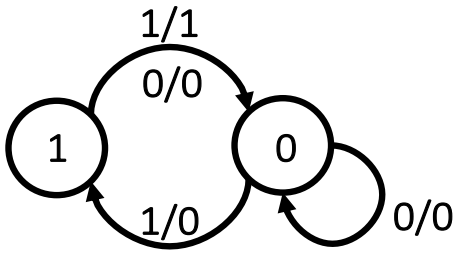
SEQUENTIAL EQUIVALENCE (MULTIPLE RESET STATES)



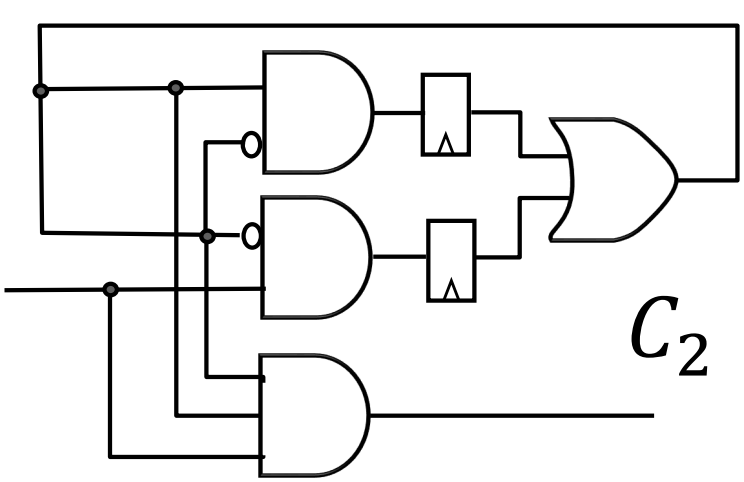
?



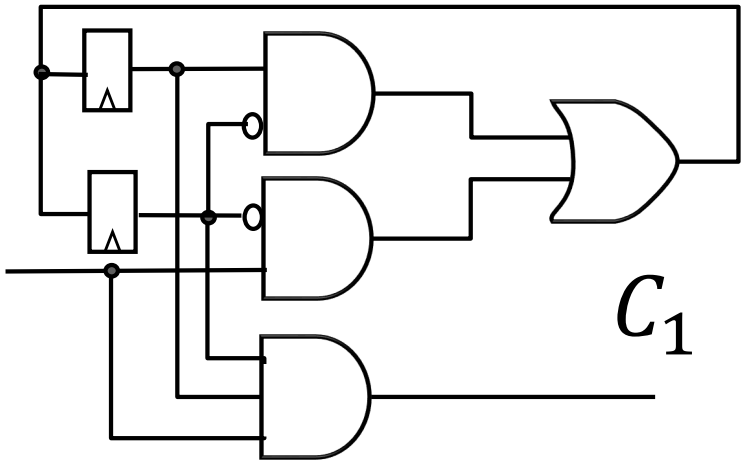


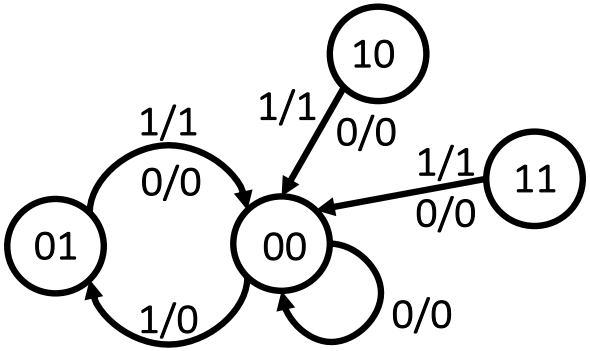


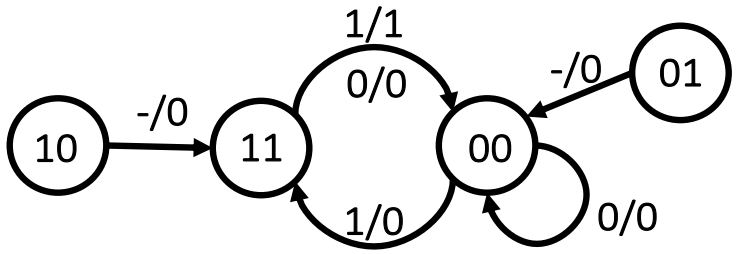
SEQUENTIAL EQUIVALENCE (MULTIPLE RESET STATES)



?

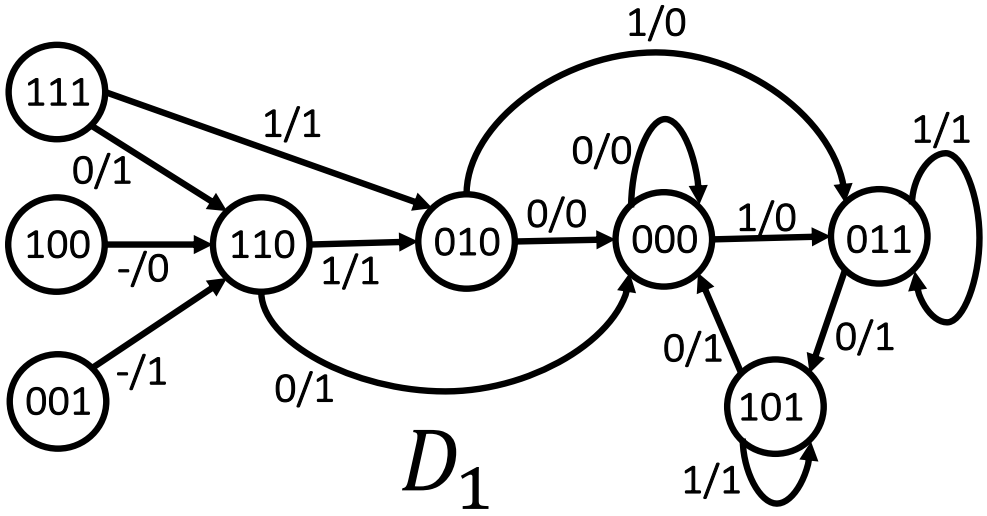


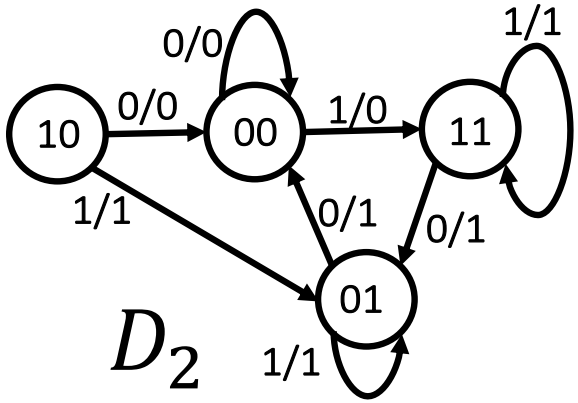




NVIDIA

SAFE REPLACEMENT

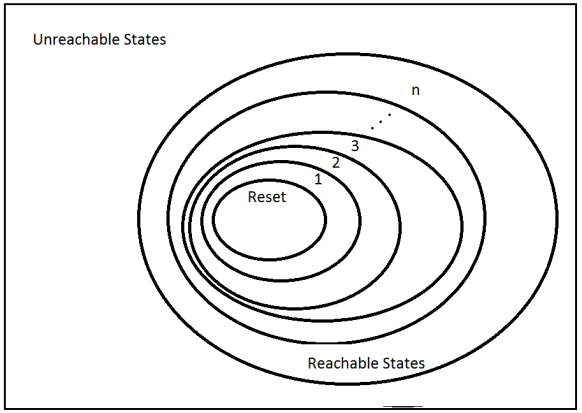




- Is D_2 a safe replacement of D_1, i.e. $D_2 \preccurlyeq D_1$?
- Is $D_1 \preccurlyeq D_2$?

REACHABLE AND UNREACHABLE STATES

- Reachable states are those that lie on at least one trace from a reset state
- Unreachable states can never be reached from any trace from a reset state
- State space is partitioned into reachable and unreachable sets



REACHABLE STATES

The behavior of the design from unreachable states is not relevant to the correctness of a property

Finding the set of reachable states is hard for many designs (2^p possible states)

=> state space explosion problem!

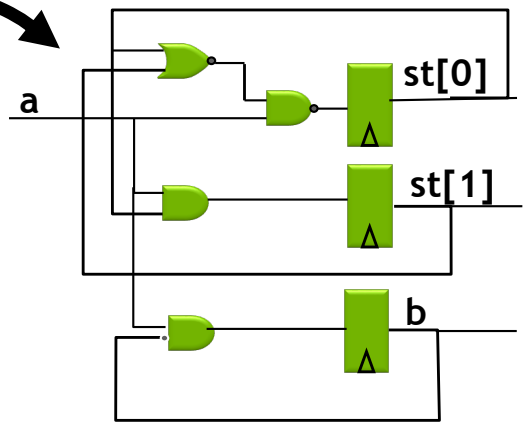
STATE SPACE EXPLOSION

RTL property: **"(st == 2'b01) |-> ~b"**

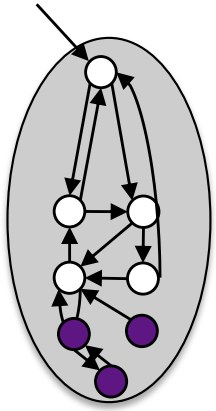
RTL

```
input a;
reg b;
reg [1:0] st;

always @(posedge clk or negedge rst)
  if (~rst) st <= 2'b00;
  else case( st )
        2'b00:  if (~a) st <= 2'b01;
        2'b01:  st <= 2'b10;
        2'b10:  if (a) st <= 2'b00;
        endcase

always @(posedge clk or negedge rst)
  if (~rst) b <= 1'b0;
  else if (~a | b) b <= 1'b0;
  else b <= 1'b1;
```

Netlist



STG



State (3-bit) = {st,b}

2^3 = 8
2^{10} = 1,024
2^{20} = 1,048,576
2^{30} = 1,073,741,824

FV complexity comes from state space explosion

NVIDIA

STATE SPACE SEARCH

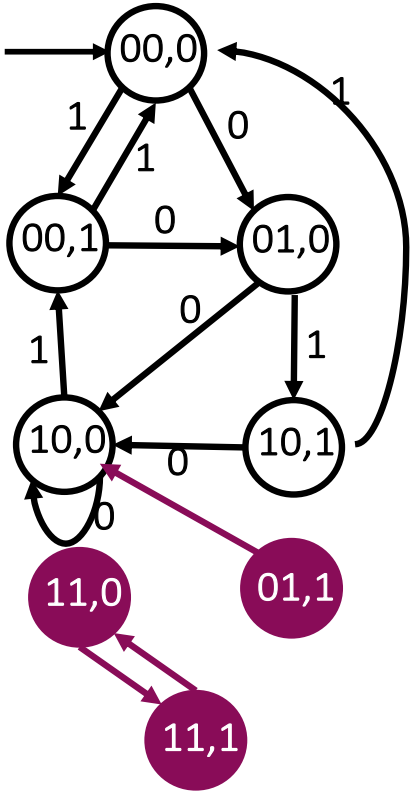
State (3-bit) = {st,b}

1. Reached set = {000}
2. Reached set = {000,001,010}
3. Reached set = {000,001,010,100,101}
4. Reached set = {000,001,010,100,101}

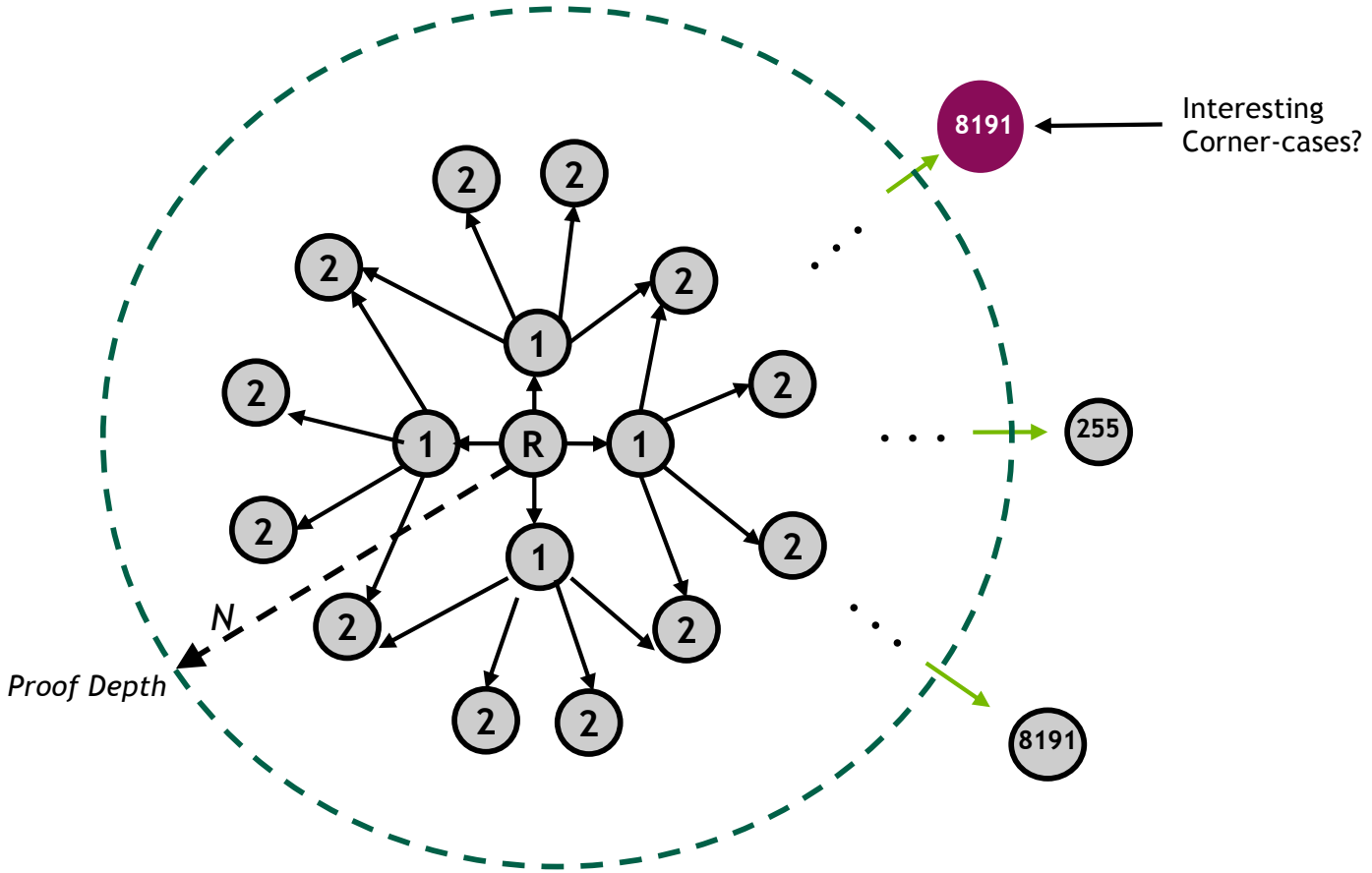
Unreachable set = {011, 110, 111}

RTL assertion: "(st == 2'b01) |-> !b"
Only one bad state: 011

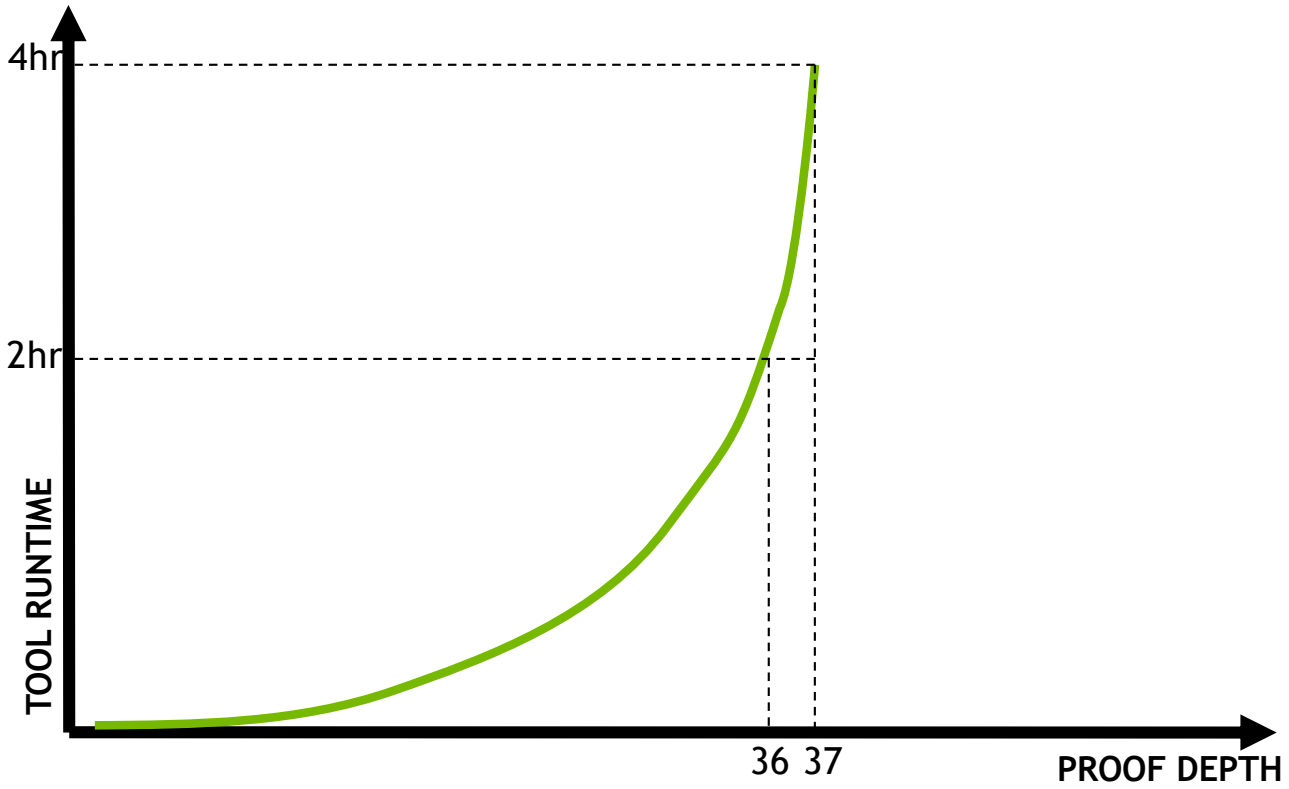
⇒ Property is true on the design



DIAMETERS OF STG'S ARE LARGE



CAN'T FIGHT THE EXPONENTIAL COMPLEXITY!



HOW HARD IS MODEL CHECKING?

- Logics
  - LTL (Linear Time Logic), Pnueli, 1977
    - Temporal logic with “eventually” and “always” (later with “next” and “until”)
  - CTL (Computation Tree Logic), Clarke and Emerson, 1981
    - Branching-time logic, with A (along all paths) and E (along some path) operators
- Upper bounds
  - CTL property of size n can be model checked on structure of size m in time $m \cdot n$ (Clarke and Emerson, 1981)
  - LTL property of size n can be model checked on structure of size m in time $m \cdot 2^{O(n)}$ (Lichtenstein and Pnueli, 1985)
    - Automata-theoretic construction by Vardi and Wolper, 1986
    - Vardi and Wolper also established this bound as the lower bound in *m* and *n*

- But how big are these structures (in hardware designs)???

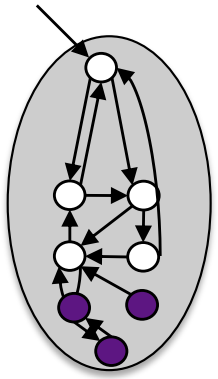
HARDER THAN COMBINATIONAL-CIRCUIT SAT

So, how hard is model checking in the size of the Verilog programs (not structure)?

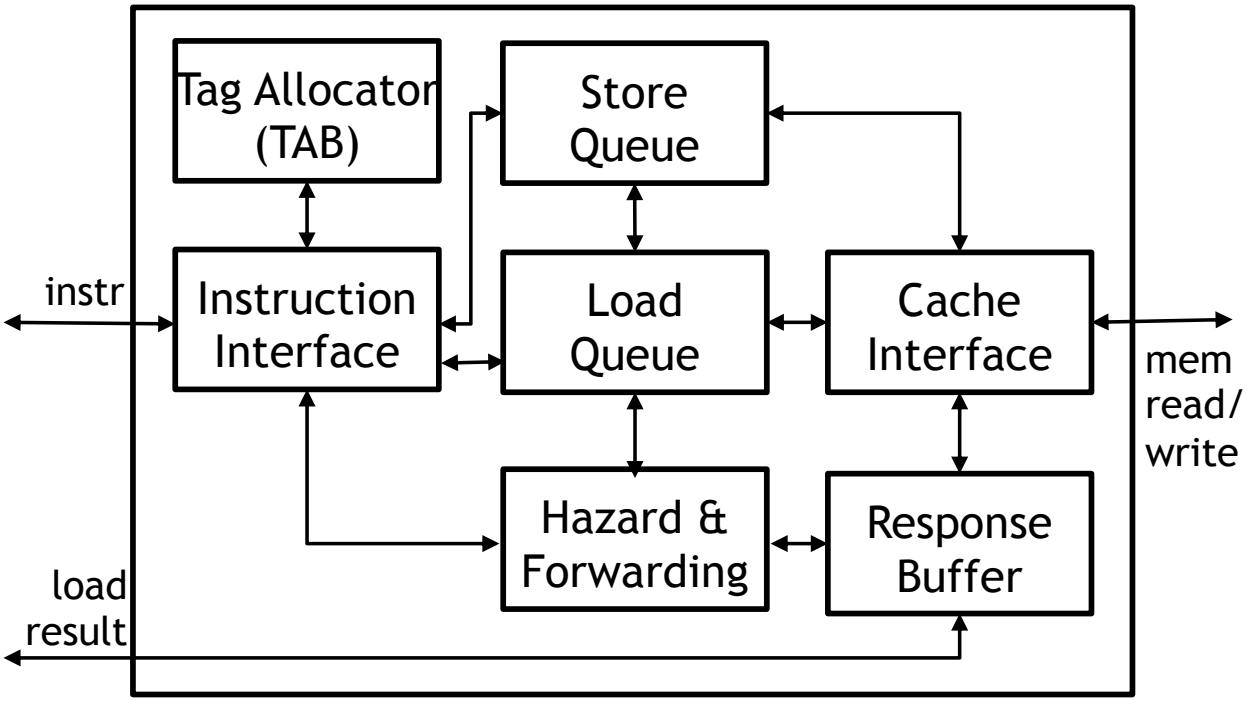
Adnan Aziz, V. Singhal, Robert Brayton, 1993

In this report we carry out a computational complexity analysis of a simple model of concurrency consisting of interacting finite state machines with fairness constraints (IFSMs). This model is based on specification languages used for system specification by actual formal verification tools, and it allows compact representation of complex systems. We prove that given a property (expressed as a formula in the logic CTL), deciding if it holds of a system of IFSMs is PSPACE-complete.

- Lemma 7: It is PSPACE-complete to decide reachability.

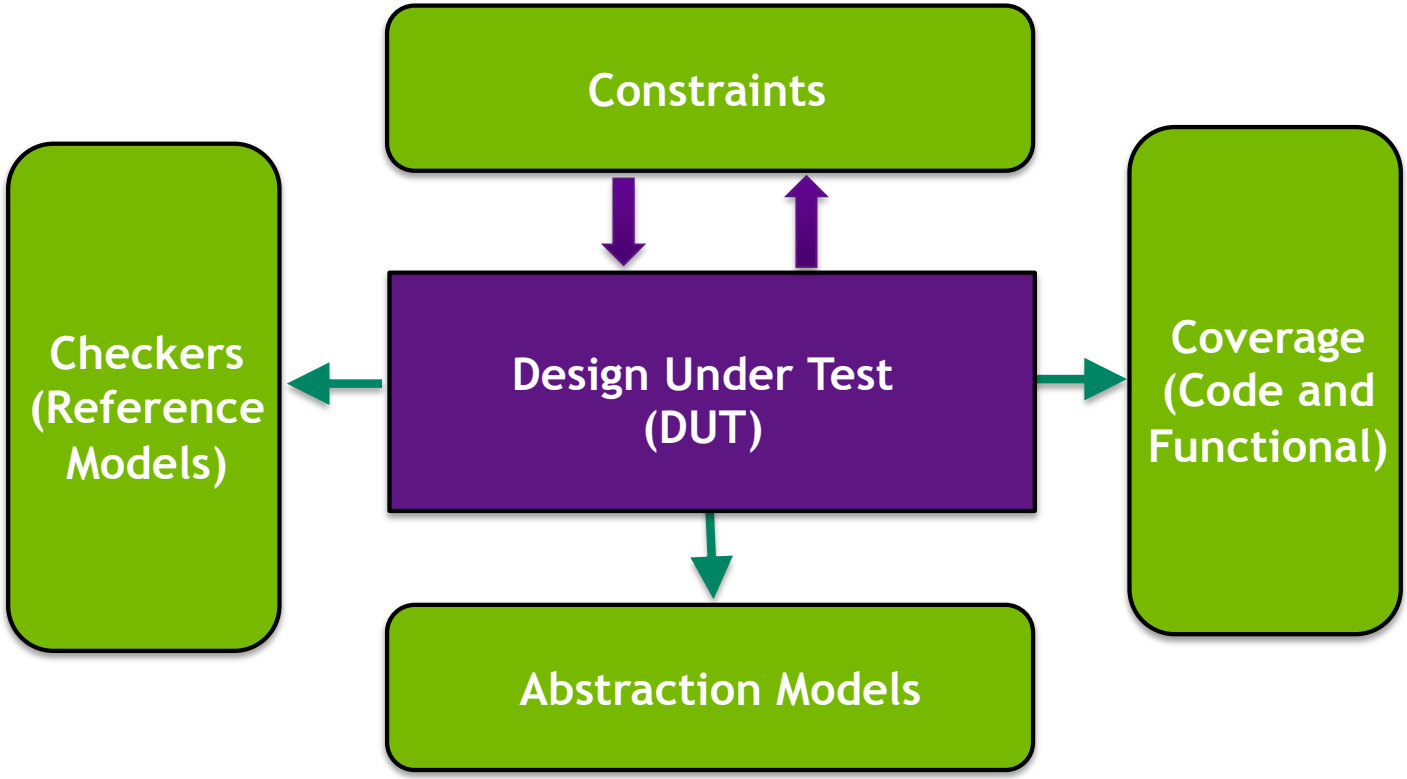


SIMPLE LOAD-STORE UNIT (SLSU)



- Decodes read (load), write (store), fence and flush ops
- Contains Load and Store queues to track pending ops
- Uses hazard and forwarding logic to optimize latency
- TAB issues unique 8-bit tag to each request going to downstream memory
- Returning data from memory is matched by tag in response buffer
- Actual design can be more complex
  - Alignment logic
  - Byte-enable optimization
  - Exceptions
  - Speculative reads
  - …

FV TESTBENCH

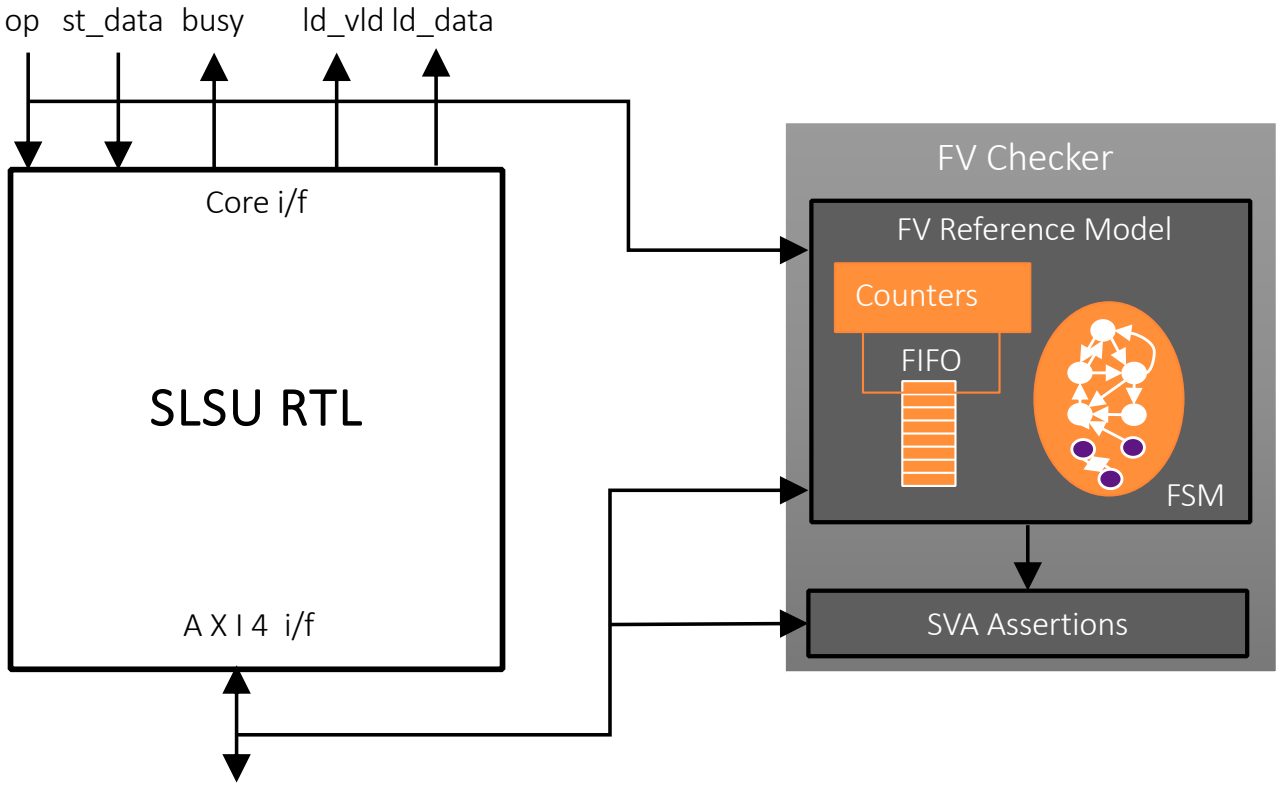


FV TESTPLAN FOR SLSU

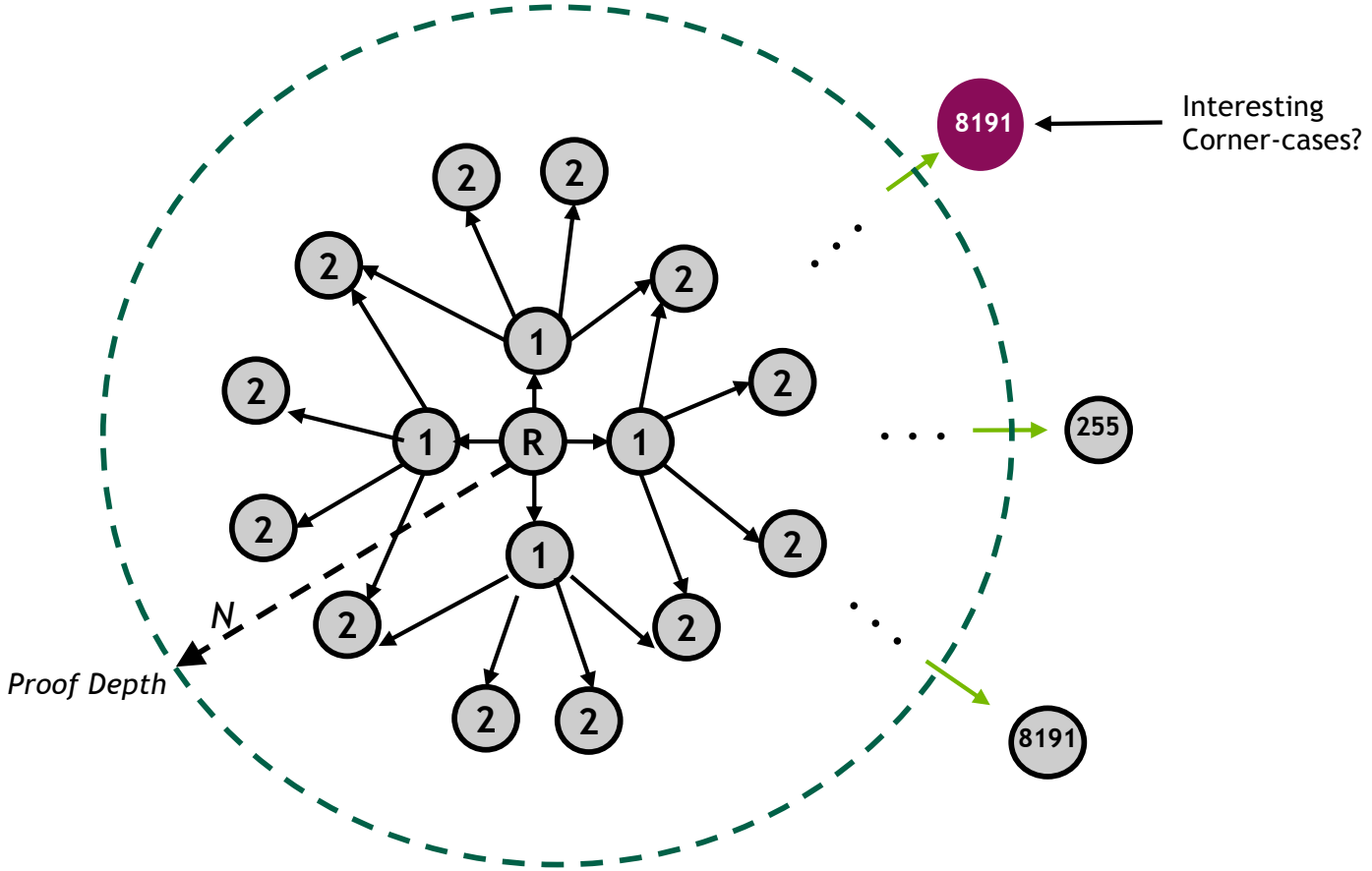
- Flow control
  - Core must not send ops if it does not have any credits (constraint)
  - After an op is processed, credits must be eventually sent back to core
  - Do not send more credits than what the core credit counter can handle
  - Memory request must be held until memory is not busy
- AXI protocol
  - LSU obeys downstream protocol rules (e.g. number of write bursts cannot exceed 256)
- Ordering and memory model
  - Younger stores must never appear before older store
  - All ops after a fence must appear after younger ops
- Data Correctness
  - Every load must return value seen by an architecturally perfect memory
  - Store data sent to memory must match the in-order architectural store data
- Tag uniqueness
  - No two outstanding memory requests share the same tag
  - Every memory response tag must match an outstanding tag (constraint)

END-TO-END CHECKERS

- ~95% of End-to-End Checker is in SV; rest is SVA
  - Developing reference model could take as much time as writing RTL



DEEP PROOF DEPTHS MAY NOT BE ACHIEVABLE



REQUIRED PROOF DEPTH

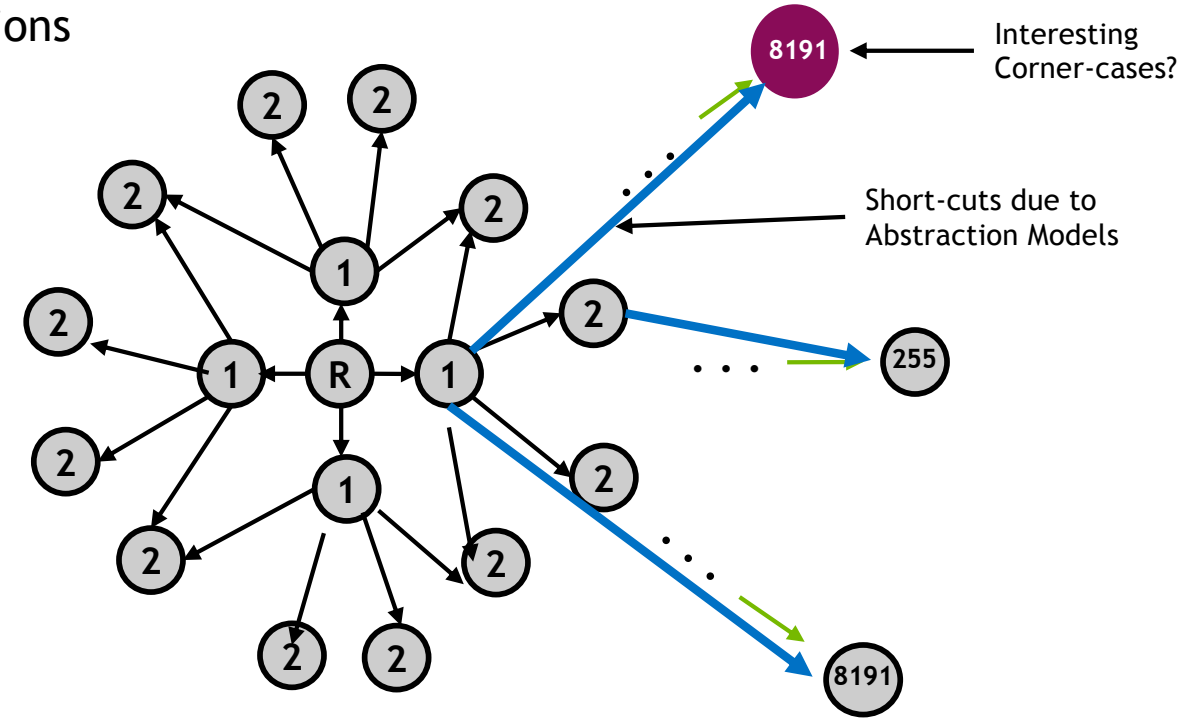
- Premise
  - A proof depth that gives us the “coverage” that “we need”
    - Similar to simulation sign-off
    - Commercial FV tools can assist in measuring coverage on code (line coverage, expression coverage)
  - A proof depth that will not miss any RTL bug
    - Bounded proof is as good as full proof, enabling formal sign-off

DETERMINING REQUIRED PROOF DEPTH

- Based on
  - Latency analysis of design
  - Micro-architectural analysis (with, and without designer)
  - Covers for “interesting” corner cases
  - Failures seen in bounded model checking
  - Formal coverage
  - Safety nets
    - Missed bugs found in simulation
    - Missed bugs found by bugg hunting FV engines
    - Negative testing/mutation coverage

ABSTRACTION ENGINEERING

- An abstraction results in a superset of the design behavior
  - Reduces state space
  - Adds state transitions
  - Adds Reset states

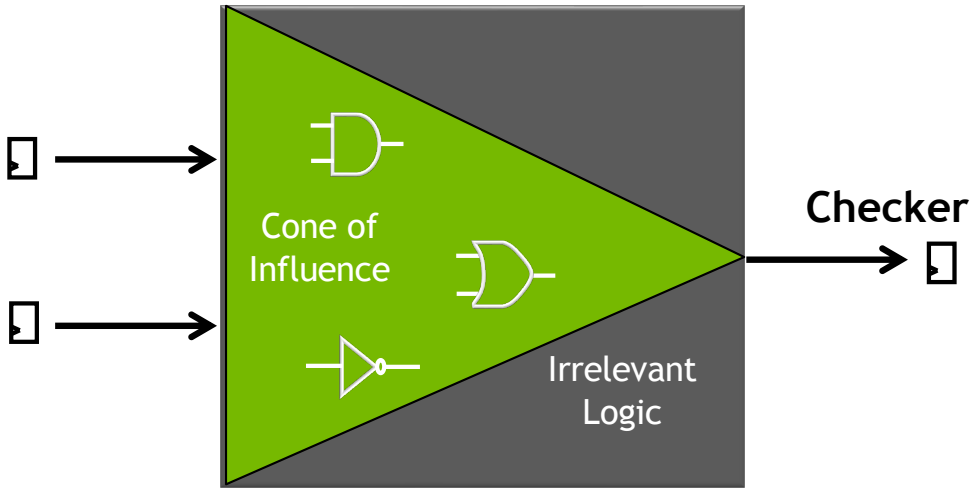


FV COMPLEXITY SOURCE 1: DEEP STATES

- Long (Input-to-Output) latency of the design
  - Common in pipelined designs
- Design structures
  - FIFO depth: a deeper FIFO takes more cycles to cover all states, e.g., the FIFO full to empty scenario will take longer number of cycles
  - Counter: an n-bit counter will take $2^n - 1$ cycles to reach its maximum value
    - Large counters will take more cycles to reach its maximum value
  - Linked list: takes n cycles to empty a linked list of n elements

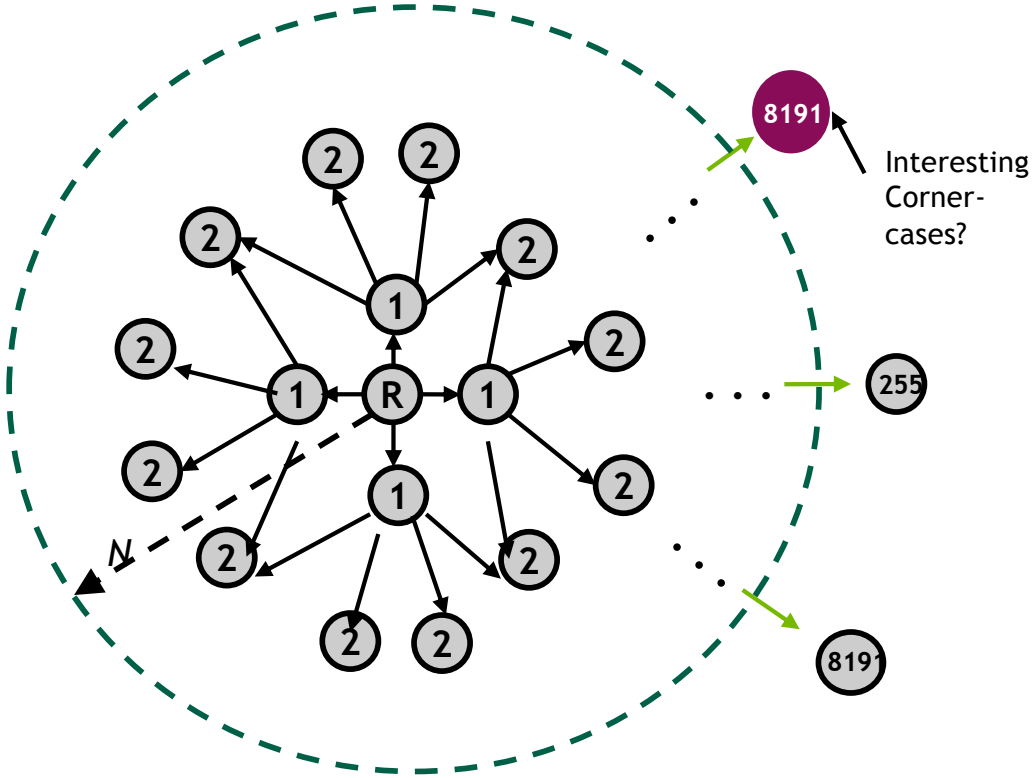
FV COMPLEXITY SOURCE 2: LARGE COI

Number of combinational gates in the Cone-of-Influence (COI) of the flops



The larger the COI, the harder each step of the state search

WHY WE NEED ABSTRACTIONS

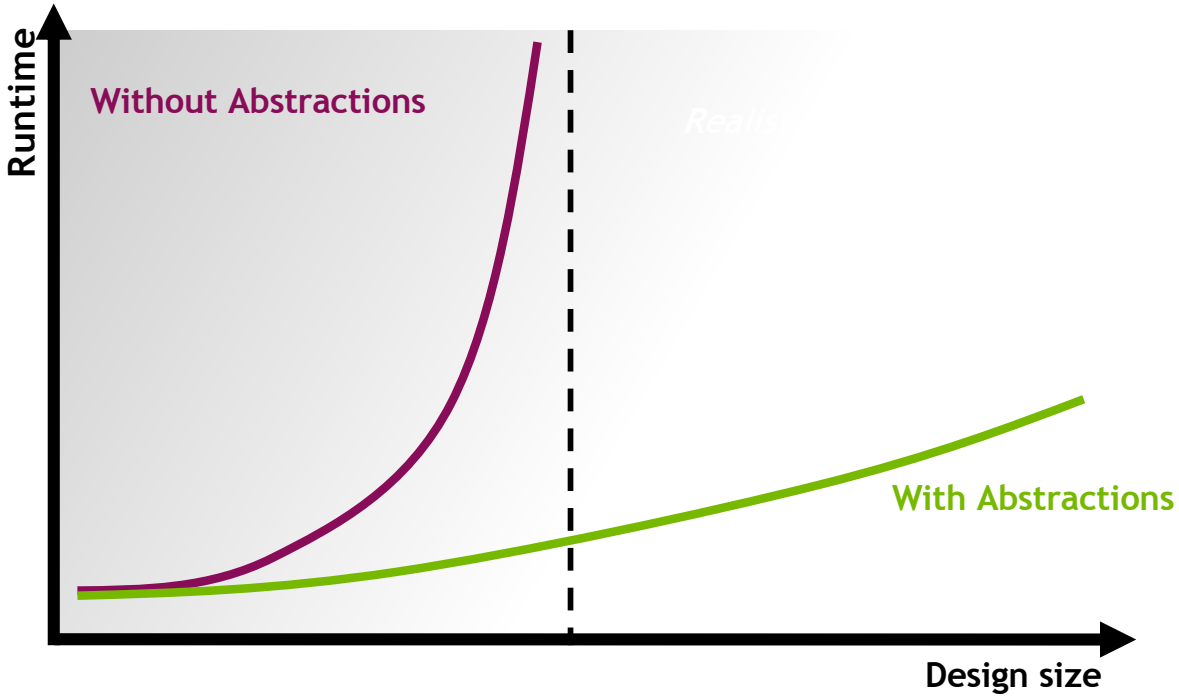


- To make sure all corner cases covered
- Abstractions can help
  1. In decreasing the depth of the corner cases which are not reachable otherwise
  2. In achieving deeper depths in same time

BENEFITS OF ABSTRACTIONS

Reduces runtime because of reduction in

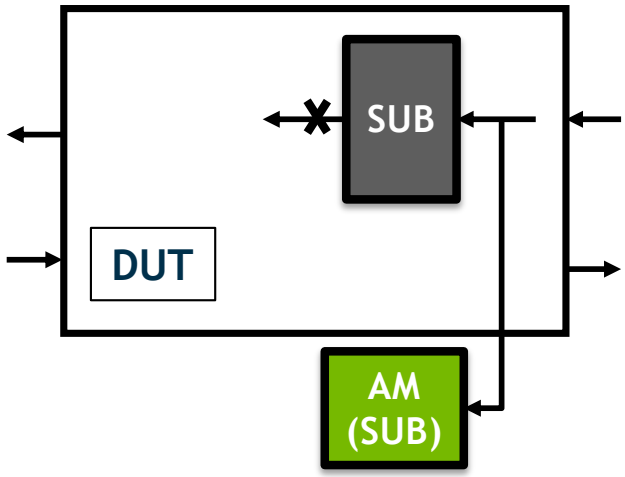
- COI
- required proof depth



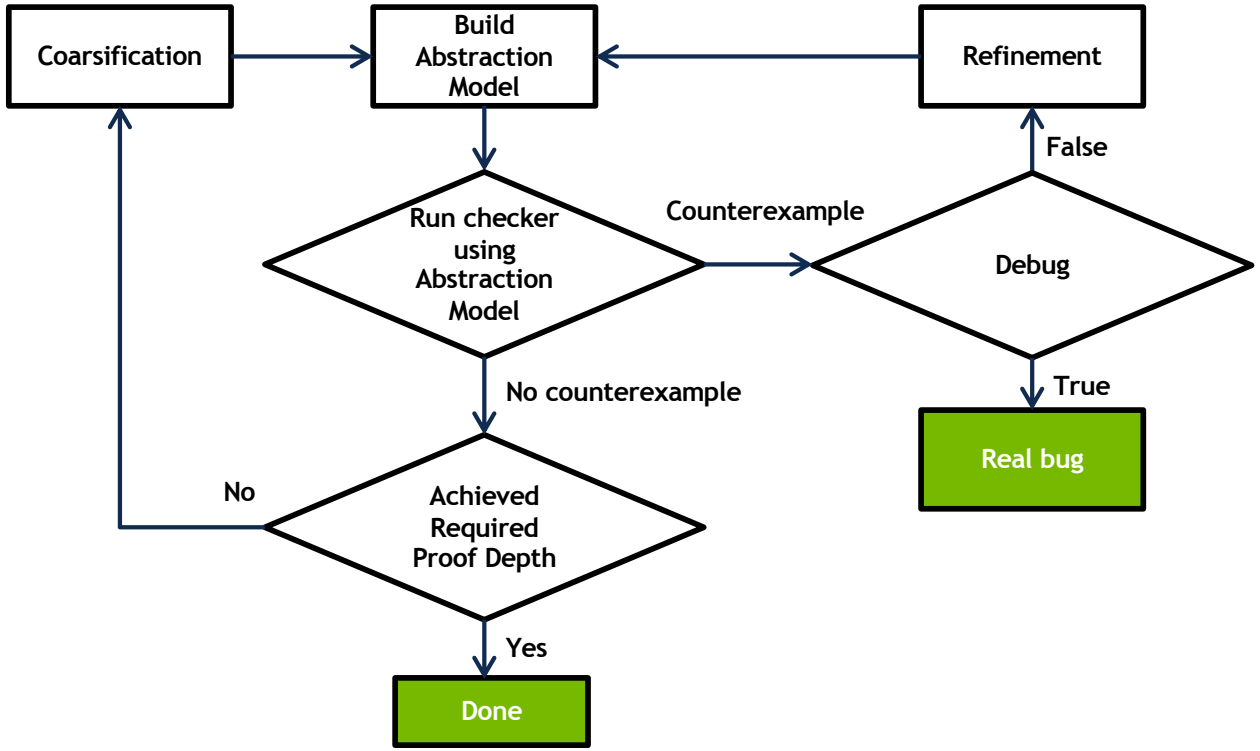
PROCESS OF BUILDING AND USING ABSTRACTIONS

- Need to analyze DUT to understand where formal complexity comes from
- Choose a boundary where we can understand behavior of boundary
- Craft design-specific AMs to overcome complexity
  - Create properties to represent abstraction
  - Cut-point RTL at the boundary
  - Use properties as constraints

- Goals of an AM
  - Should be effective at reducing complexity
    - Should be coarse enough
  - Should not give false failures
    - Should be fine enough

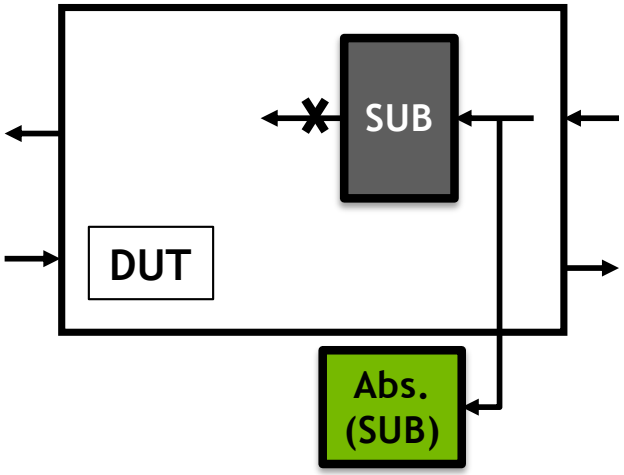


ITERATION FLOW DIAGRAM



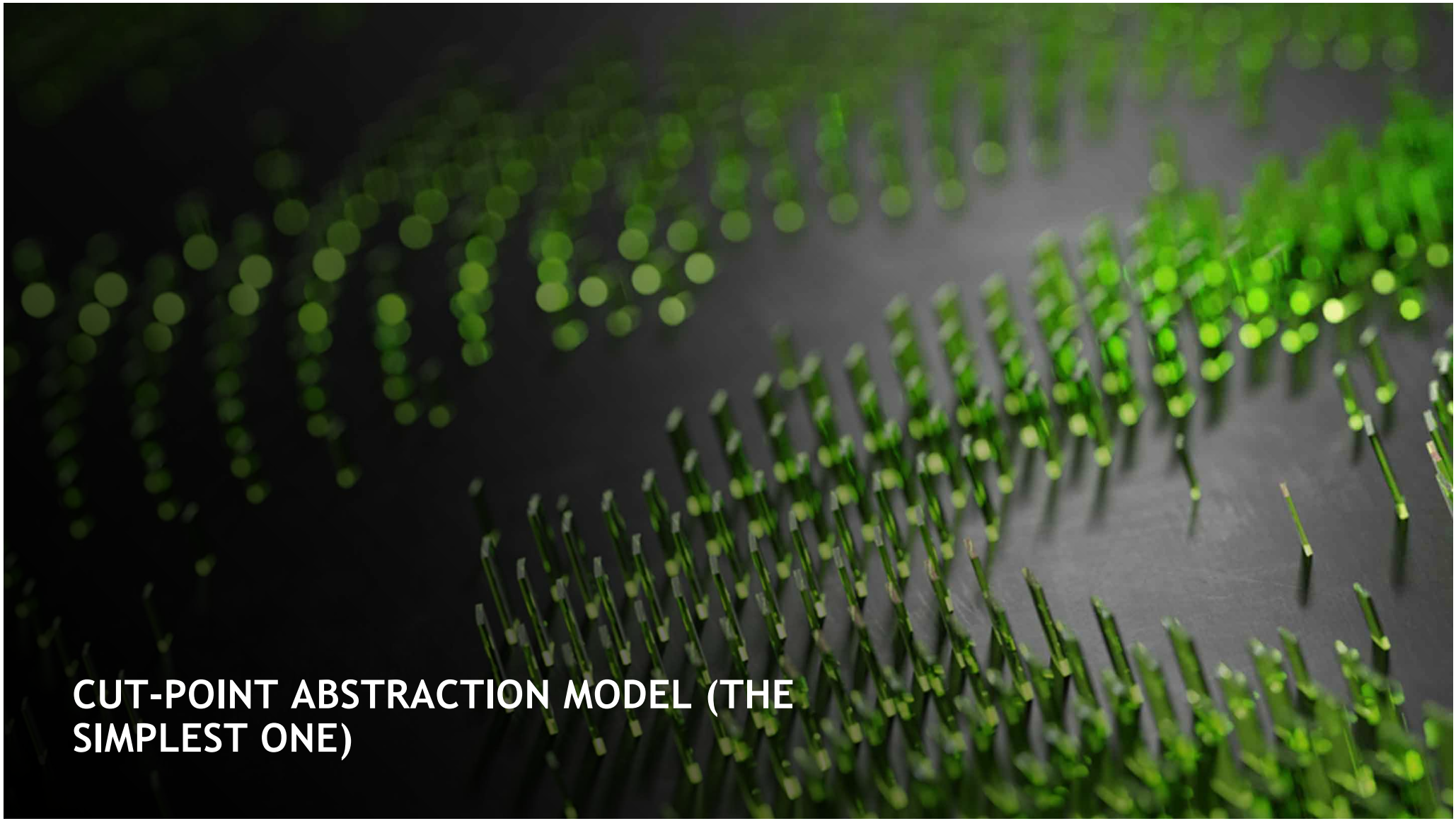
USING & PROVING CORRECTNESS OF ABSTRACTIONS

- Using
  - Add cut-points
  - Bind Abstraction Model to design
  - Assume properties (use as constraints)

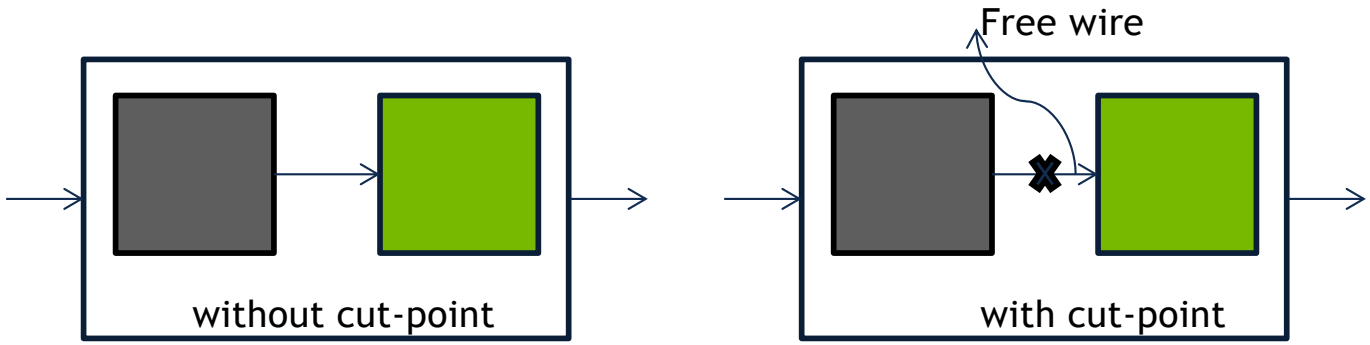


- Proving
  - Prove previously assumed properties

Small model, so deep proof depth is reachable!

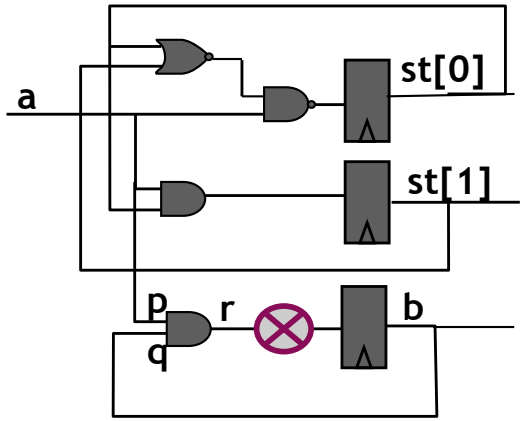

CUT-POINT ABSTRACTION MODEL (THE
SIMPLEST ONE)

CUT-POINT ABSTRACTION



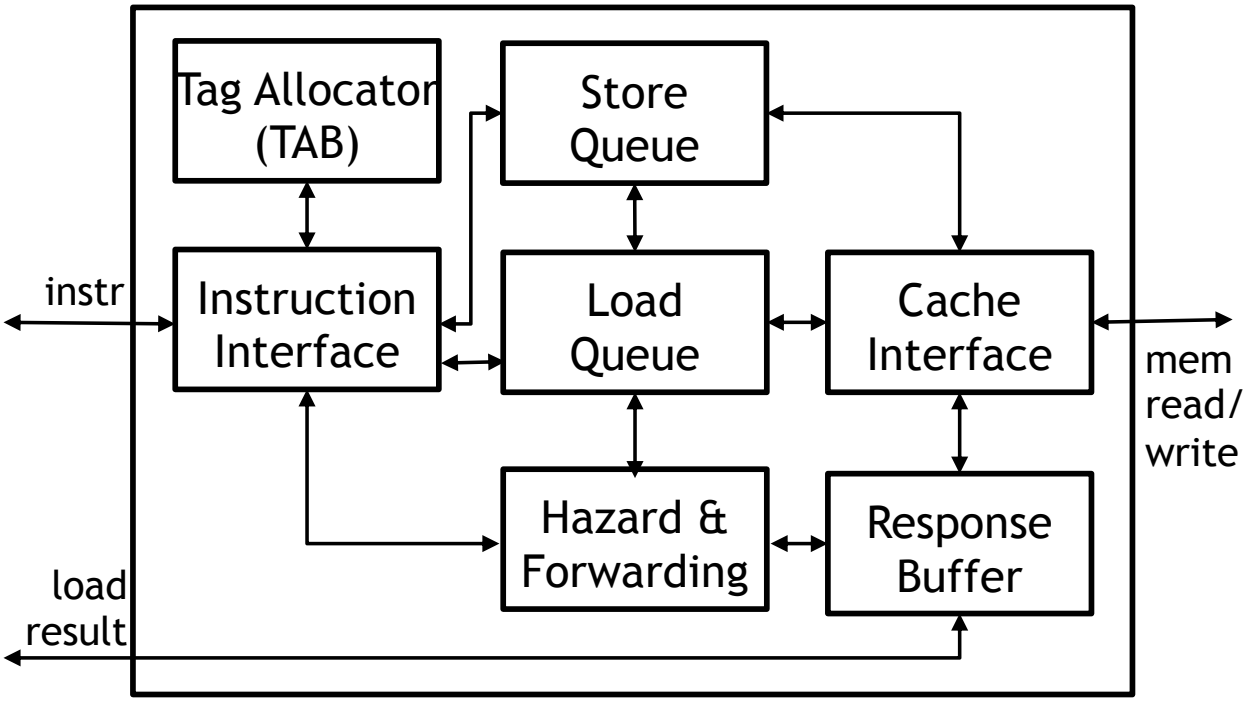
- Adds a cut-point on some signal
- Signal is free (FV can freely choose any value of that signal)
  - A superset of the design behavior without cut-points
- Logic that was driving the signal (and not used elsewhere) is not used during formal analysis
  - Reduces COI
  - Also, can reduce state-space
- In a cut-point abstraction, there are no additional constraints
  - No need to do the step to prove abstraction

CUT-POINT ABSTRACTION – EXAMPLE



- Adding a cut-point abstraction on *r* allows it to take any value on any cycle
  - E.g. *r* could be 0 even if both *p* and *q* are 1
- Has to be done judiciously
  - Too many false failures will require refinements

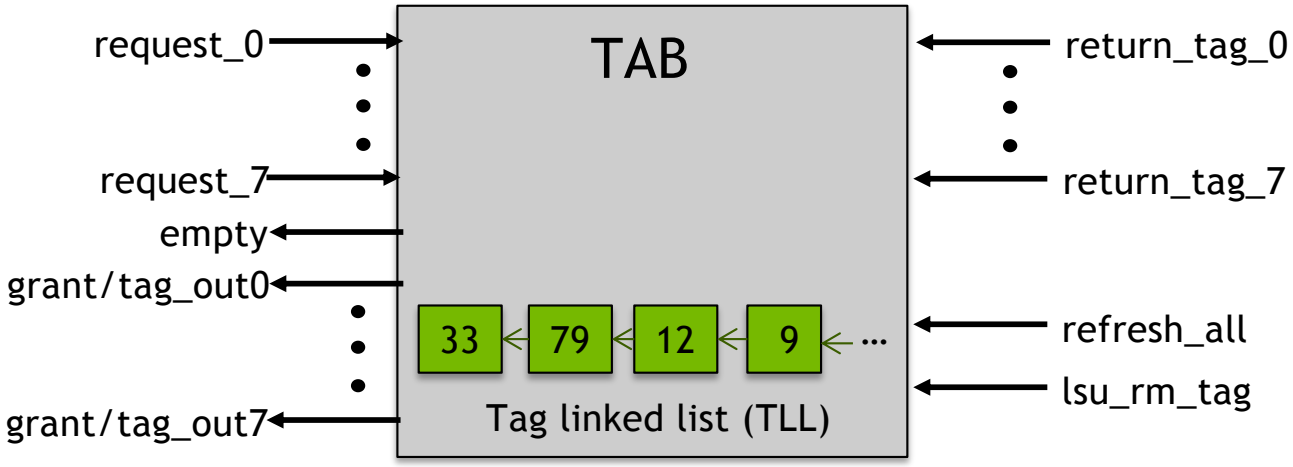
SIMPLE LOAD-STORE UNIT (LSU)



- Decodes read (load), write (store), fence and flush ops
- Contains Load and Store queues to track pending ops
- Uses hazard and forwarding logic to optimize latency
- TAB issues unique 8-bit tag to each request going to downstream memory
- Returning data from memory is matched by tag in response buffer
- Actual design can be more complex
  - Alignment logic
  - Exceptions
  - Speculative reads
  - …

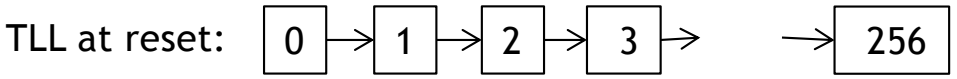
TAG ALLOCATOR BLOCK (TAB)

- Disperses unique tags to every new transaction that comes in
  - 2 requesters for 256 tags with 1 grant in next cycle
  - TAB stores the tags in a large linked list of size 256
  - 2 tag returns in single cycle; LSU can refresh all tags or remove a tag



TAB ABSTRACTION

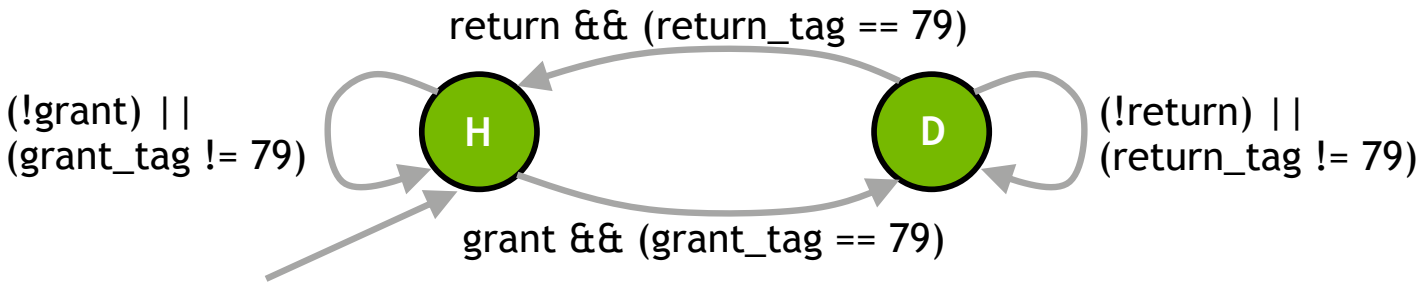
- Due to large size of linked list, proving SLDU end-to-end properties of SLSU is harder
  - Requires 256 cycles to reach the scenario of empty linked list



256 cycles to cover (empty = 1)!

- To solve this complexity problem, we need Abstraction Model of TAB
  - Mark one of the 256 tag values as special, e.g. value 79
  - Abstraction Model of TAB based on two states
    - H: TAB Has special tag
    - D: TAB Doesn't have special tag

TAB ABSTRACTION



Properties on TAB outputs (P_{TAB}) used as constraints when TAB is replaced by its abstraction

1. request |-> ##1 (empty || grant)
2. (state == H) |-> (!empty)
3. ((state == D) && (grant)) |-> (grant_tag != 79)

Properties on TAB inputs (P_{SYS}) used as checkers when TAB is replaced by its abstraction

1. ((state == H) && return) |-> (return_tag != 79)
2. (state == D) |-> *"tag 79 is eventually returned"*

TAB ABSTRACTION

- Advantages
  - Reduces COI of the checker, and therefore reduces state-space
  - Reduces required proof-depth for the checker

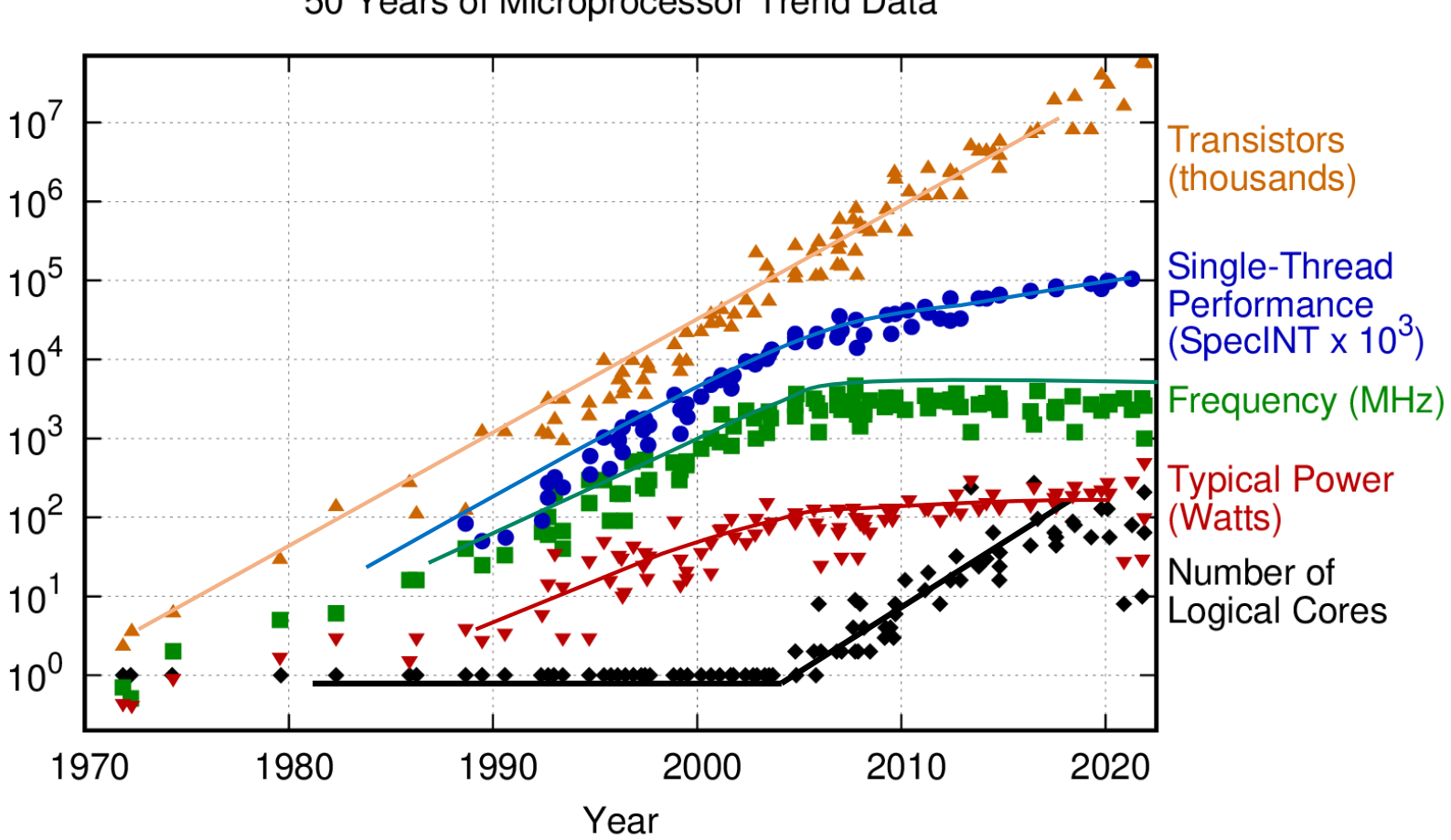
- Proving Tag Allocator AM against TAB RTL
  - Reverse the usage of properties on TAB outputs and inputs, i.e.
    - Properties on TAB outputs (P_{TAB}) used as checkers and properties on TAB inputs (P_{SYS}) used as constraints
  - Can be sequentially long, but on a tiny DUT


QUESTIONS?
NVIDIA

AGENDA

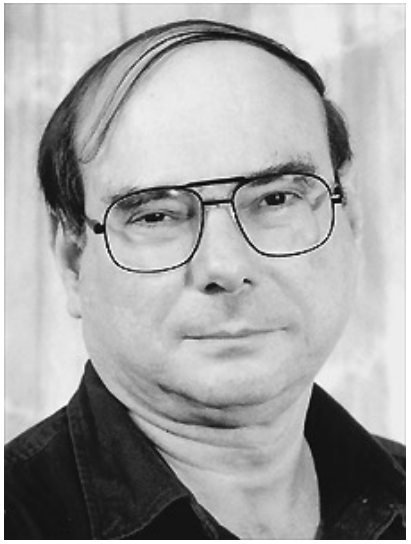
- Tribute to Bob Brayton
- Chip design process
- Important of verification in chip design
- Logic circuits in chip design
- History of FV tools
- FV methodology
  - Required Proof Depths
  - Abstraction

TRANSISTOR COUNTS AND PIVOT TO PARALLELISM



Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2021 by K. Rupp

THE TEMPORAL LOGIC OF PROGRAMS



Amir Pnueli, 1977 (> 7,500 citations)

A unified approach to program verification is suggested, which applies to both sequential and parallel programs. The main proof method suggested is that of temporal reasoning in which the time dependence of events is the basic concept. Two formal systems are presented. One forms a formalization of the method of intermittent assertions, while the other is an adaptation of the tense logic system Kb and is particularly suitable for reasoning about concurrent programs.

- Turing Award 1996 citation:
  - For seminal work introducing temporal logic into computing science and for outstanding contributions to program and system verification.

(Slide from Moshe)

AUTOMATIC VERIFICATION OF FINITE-STATE CONCURRENT SYSTEMS

Ed Clarke, Allen Emerson, 1981, JP Queille, Joseph Sifakis, 1982
(> 7,700 citations)

We give an efficient procedure for verifying that a finite-state concurrent system meets a specification expressed in a branching-time temporal logic. We argue that this technique can provide a practical alternative to manual proof construction or use of a mechanical theorem prover for verifying many finite-state concurrent systems.

- Turing Award 2007 citation:
  - For their role in developing Model Checking into a highly effective verification technology that is widely adopted in the hardware and software industries.

(Slide from Moshe)

AN AUTOMATA-THEORETIC APPROACH TO AUTOMATIC PROGRAM VERIFICATION

Moshe Vardi, Pierre Wolper*, 1986 (> 2,300 citations)

We describe an automata-theoretic approach to the automatic verification of concurrent finite-state programs by model checking. The basic idea underlying this approach is that for any temporal formula we can construct an automaton that accepts precisely the computations that satisfy the formula. The model-checking algorithm that results from this approach is much simpler and cleaner than tableau-based algorithms. We use this approach to extend model checking to probabilistic concurrent finite-state programs. concurrent finite-state programs.

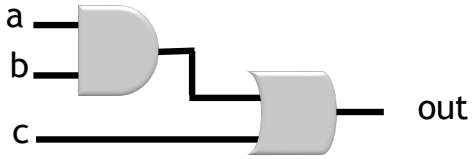
- Gödel Prize 2000 citation:
  - This paper is a reworking and extension of a conference contribution of FOCS'83, which has become a major reference in the automata-theoretic approach to temporal logic.

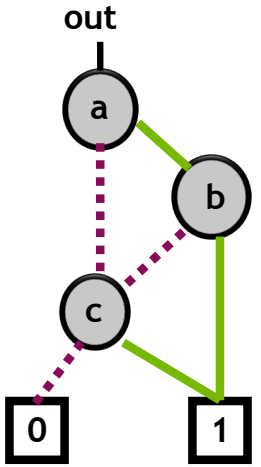
* Expressing interest properties of programs in propositional temporal logic. Wolper. (> 400 citations)

NVIDIA

GRAPH ALGORITHMS FOR BOOLEAN FUNCTION MANIPULATION (BDD'S)

Randy Bryant, 1986 (> 12,000 citations)





- Phil Kaufman Award 2009 citation:
  - Dr. Bryant received this award for his seminal technological breakthroughs in the area of formal verification. Dr. Bryant's research focuses on methods for formally verifying digital hardware and some forms of software. Notably, he developed efficient algorithms based on ordered binary decision diagrams (OBDDs) to manipulate the logic functions that form the basis for computer designs. His work revolutionized the field, enabling reasoning about large-scale circuit designs for the first time.

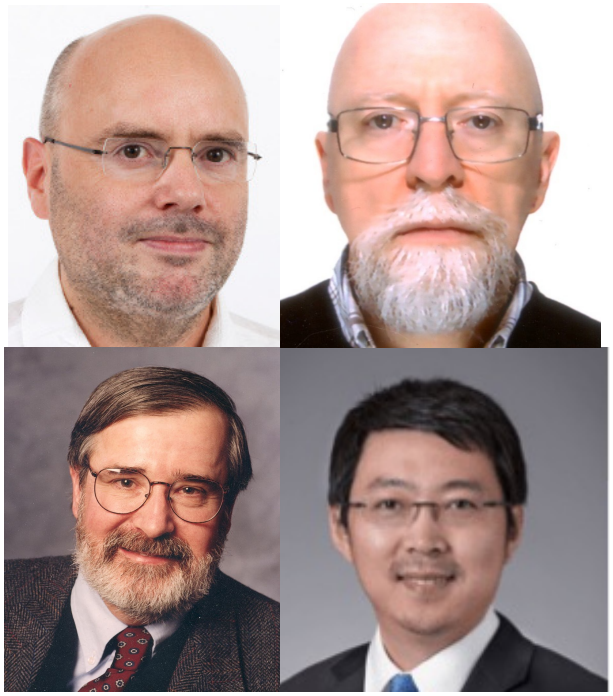
SYMBOLIC MODEL CHECKING (THE SMV SYSTEM, CA. 1987)

Ken McMillan, 1993 PhD thesis (> 6,000 citations)

Finite state models of concurrent systems grow exponentially as the number of components of the system increases. This is known widely as the state explosion problem in automatic verification and has limited finite state verification methods to small systems. To avoid this problem, a method called symbolic model checking is proposed and studied. This method avoids building a state graph by using Boolean formulas to represent sets and relations. A variety of properties characterized by least and greatest fixed points can be verified purely by manipulations of these formulas using Ordered BDDs.

- LICS Test of Time Award 2010 citation (1990 paper by Burch/Clarke/McMillan/Dill/Hwang)
  - This paper revolutionized model checking. Through its symbolic representation of the state space using Randy Bryant's BDDs and its careful analysis of several forms of model checking problems, backed up by empirical results, it provided a first convincing attack on the verification of large-state systems.

SYMBOLIC MODEL CHECKING WITHOUT BDD'S (BMC)



A. Biere, A. Cimatti, E. Clarke, Y. Zhu, 1999 (> 3,200 citations)

Symbolic Model Checking has proven to be a powerful technique for the verification of reactive systems. BDDs have traditionally been used as a symbolic representation of the system. In this paper we show how boolean decision procedures, like Stålmarck's Method or the Davis & Putnam Procedure, can replace BDDs. This new technique avoids the space blow up of BDDs, generates counterexamples much faster, and sometimes speeds up the verification. In addition, it produces counterexamples of minimal length.

- CAV 2018 citation (along with Kroening and Lerda)
  - For their outstanding contribution to the enhancement and scalability of model checking by introducing Bounded Model Checking based on Boolean Satisfiability (SAT) for hardware (BMC) and software (CBMC).