FMCAD 2022

Error Correction Code Algorithm and Implementation Verification Using Symbolic Representations

Aarti Gupta, Roope Kaivola, Mihir Parang Mehta, Vaibhav Singh

intel®

FMCAD 2022

Error Correction Code Algorithm and Implementation Verification Using Symbolic Representations

Aarti Gupta, Roope Kaivola, Mihir Parang Mehta, Vaibhav Singh

Background

- Data corruption due to Soft Errors (random, non-recurring bit errors in memory devices) can lead to system failures.
- Error Correction Codes (ECCs) were introduced in memory circuits to make systems fault-tolerant.
- ECCs cannot be verified fully using Traditional Dynamic Simulation, and Formal Methods are needed.

REUTERS

OCTOBER 14, 2008

Australia blames computer glitch for Airbus mishap

THE WALL STREET JOURNAL.

For a few hours in 2003, on national election day, Maria Vindevoghel thought she had started a Belgian revolution. Then she found out that a binary-code malfunction caused by a cosmic ray had given her Communist Party 4,096 extra votes in Schaerbeek, a Brussels precinct.

SUPERCOMPUTING'S MONSTER IN THE CLOSET

By Al Geist

A high-profile example affected what was the second fastest supercomputer in the world in 2002, a machine called ASCI Q at Los Alamos National Laboratory. When it was first installed at the New Mexico lab, this computer couldn't run more than an hour or so without crashing.

Error Correction Codes

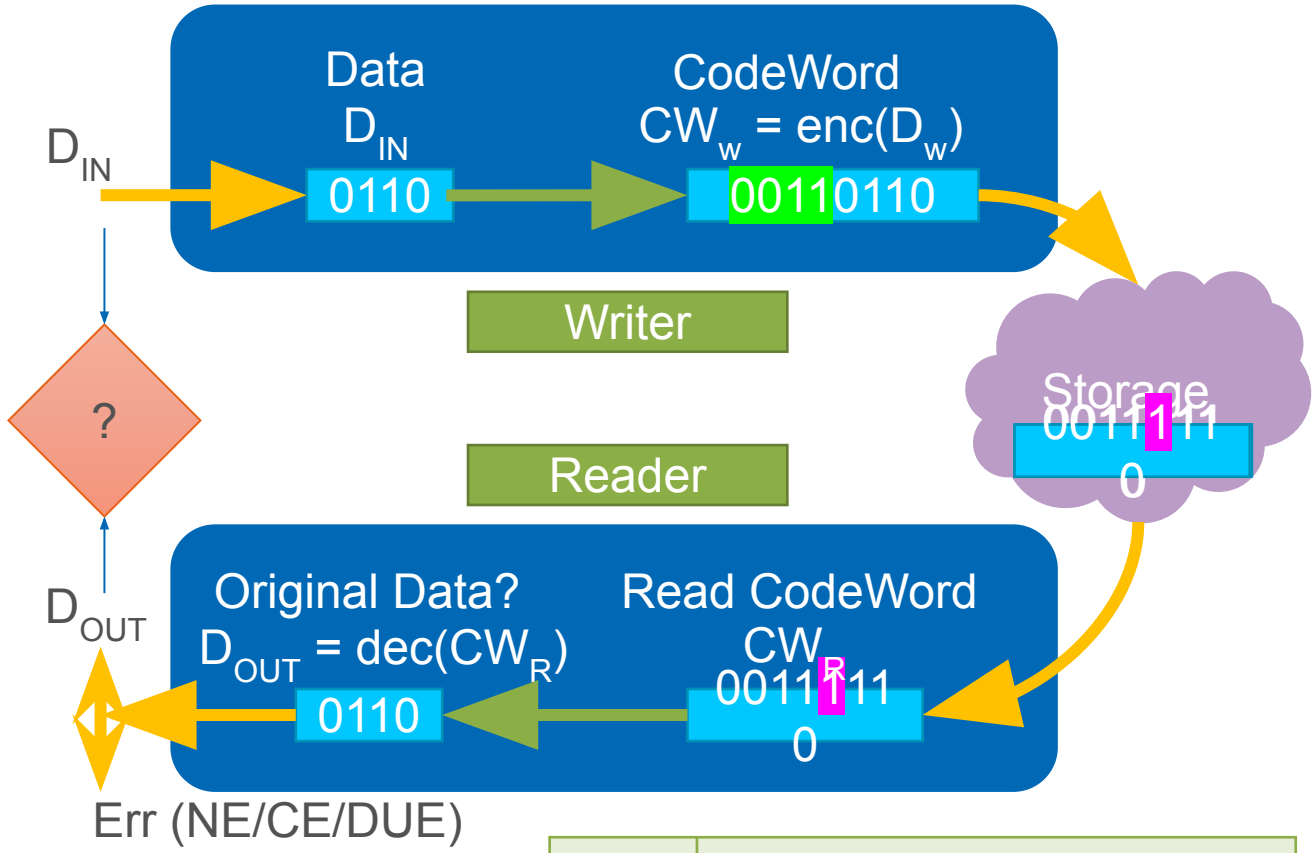
In a Nutshell

Error Correction Codes

An error correction code (ECC) is an encoding scheme that transmits messages as binary numbers, in such a way that the message can be recovered even if some bits are erroneously flipped.

- Parity Schemes
- Hamming Codes
- BCH Codes
- Golay Codes
- Reed-Solomon Codes
- Customized Algorithms

Examples of ECC Algorithms

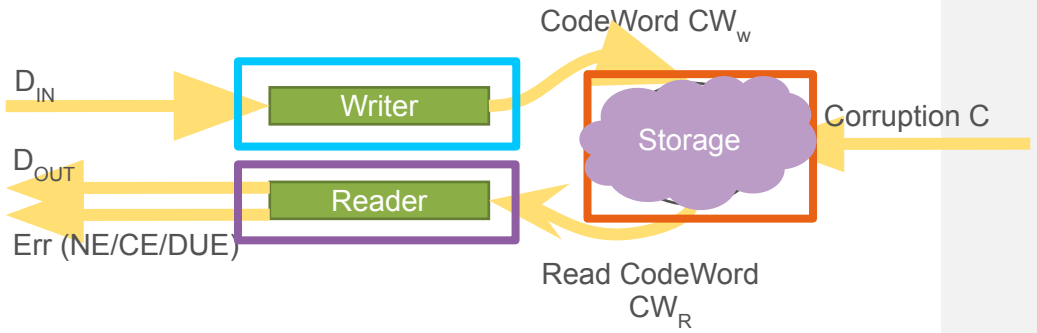


NE	No Error
CE	Correctable Error
DUE	Detectable but Uncorrectable Error

Error Correction Codes

Verification Challenges

ECC: Simple Example



Writer/Encoder

$$\overrightarrow{CW_w} = \overrightarrow{D} \times G = (D[0] \quad D[1] \quad D[2] \quad D[3]) \times \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \end{pmatrix}$$

Summing operation in this matrix multiplication is modulo-2 which is equivalent to digital XOR operation

Storage/Corruption

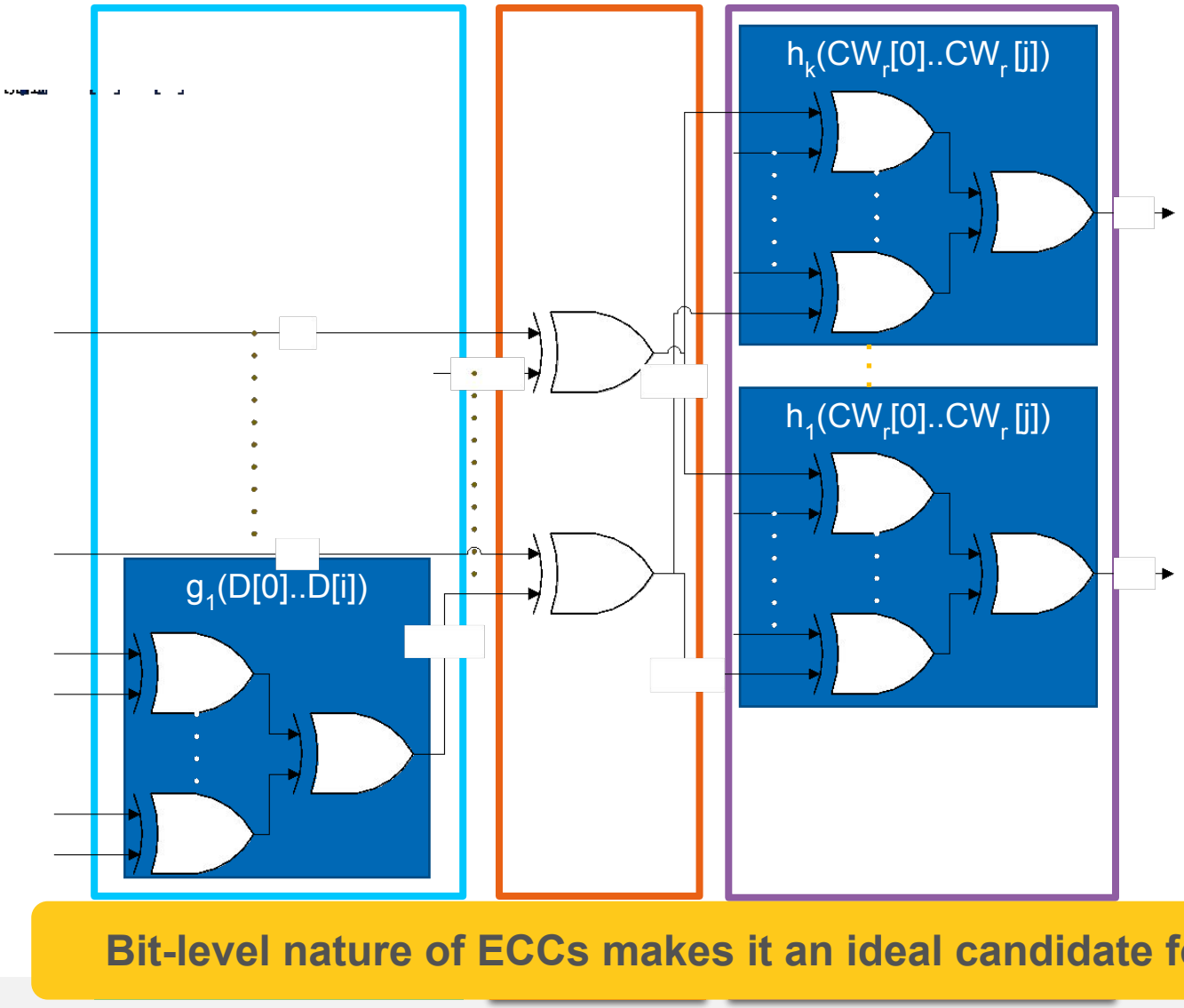
$$\overrightarrow{CW_r} = \overrightarrow{CW_w} \oplus \vec{C}$$

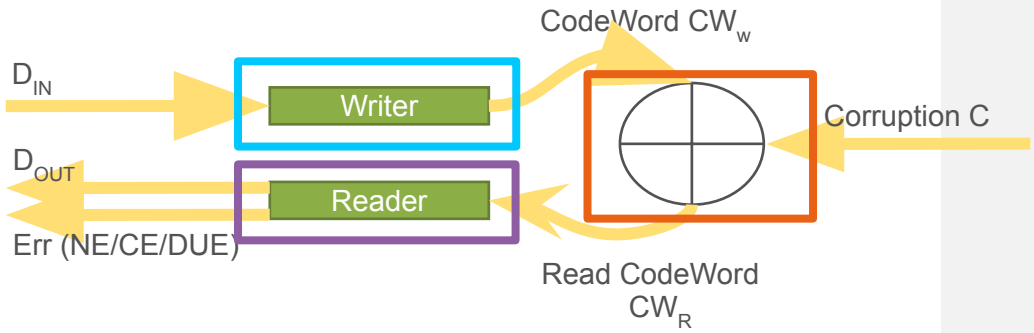
Reader/Decoder

$$\vec{S} = H \times \overrightarrow{CW_r}^T = \begin{pmatrix} 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} D[0]\oplus C[0] \\ D[1]\oplus C[1] \\ D[2]\oplus C[2] \\ D[3]\oplus C[3] \\ D[1]\oplus D[2]\oplus D[3]\oplus C[4] \\ D[0]\oplus D[2]\oplus D[3]\oplus C[5] \\ D[0]\oplus D[1]\oplus D[3]\oplus C[6] \\ D[0]\oplus D[1]\oplus D[2]\oplus C[7] \end{pmatrix}$$

Syndrome S is decoded for error presence and location

ECC: Implementation View





- Sea of XORs
- Dependent on each bit of data & corruption
- Model-checkers struggle on large designs
- Word-level engines don't help
- G & H matrices optimized frequently -> equivalence checking against a reference model impractical

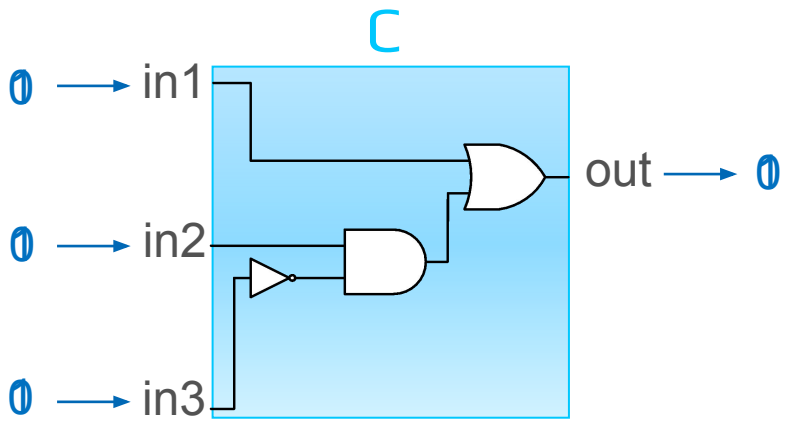
Bit-level nature of ECCs makes it an ideal candidate for Symbolic Simulation [Seger, Bryant '95]

Symbolic Simulation

A Quick Overview

Traditional Dynamic Simulation

Problem: Verify that circuit C satisfies specification S.

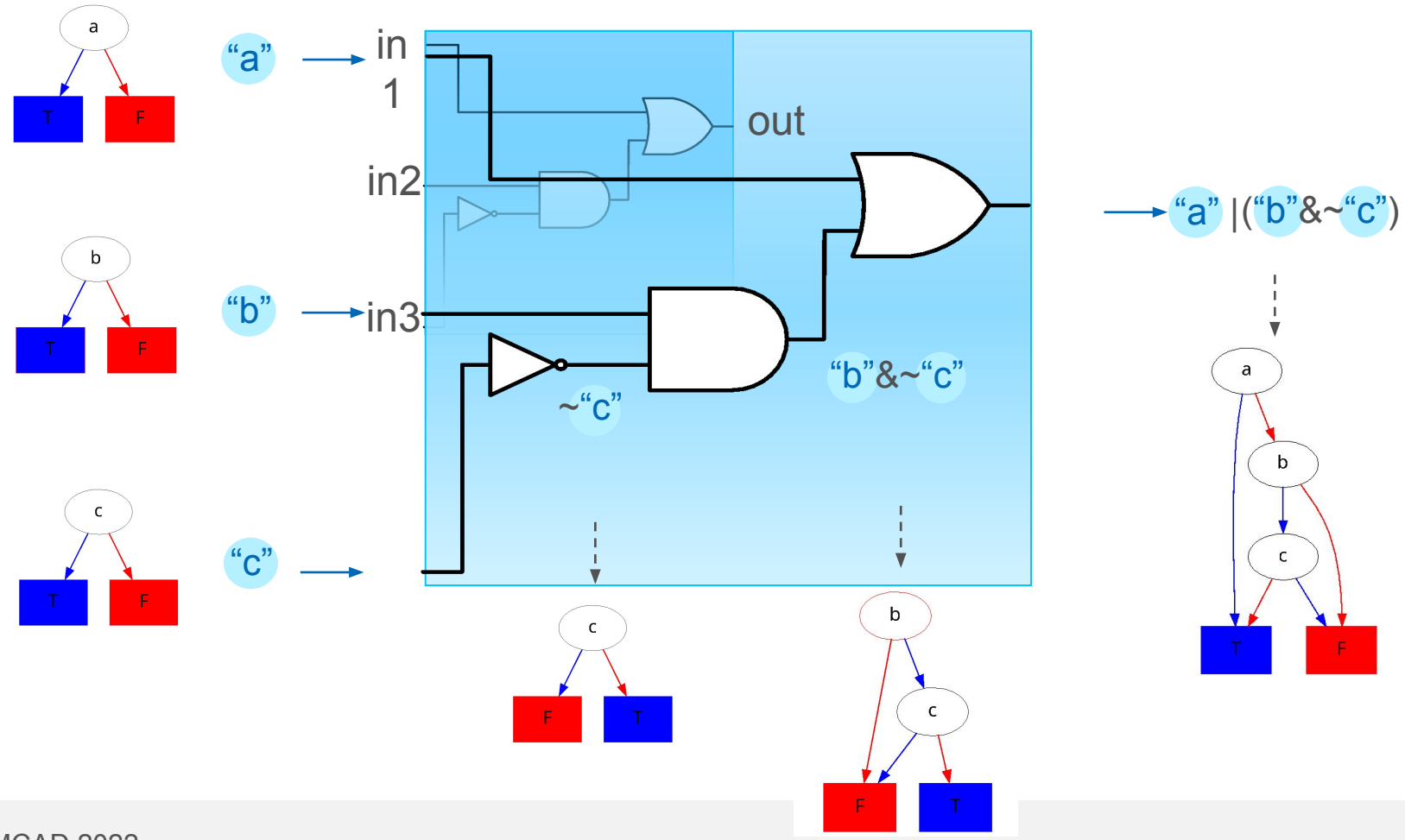


S

S (a,b,c) =
a OR (b AND NOT c
)

- Traditional simulation:
  - Inject random values and compare result to reference model
  - 2^n simulations to cover an n-bit wide logic.

Symbolic Simulation



S

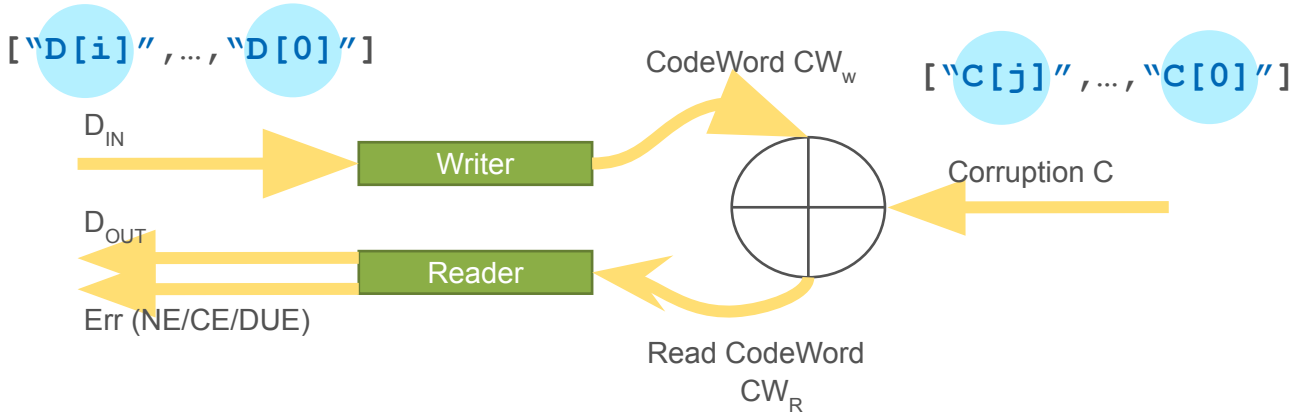
S (a,b,c) =
a OR (b AND NOT c
)

ECC Verification

Using Symbolic Simulation

ECC Symbolic Analysis

Input and Output BDDs analyzed to guarantee correctness of ECC algorithm



Generic properties

n Correction

n+1 Detection

- (Countbits(C) = 0) ⇒ NE and $D_{OUT} = D_{IN}$
- 0 < Countbits(C) <= n ⇒ CE and $D_{OUT} = D_{IN}$
- (Countbits(C) = n + 1) ⇒ DUE

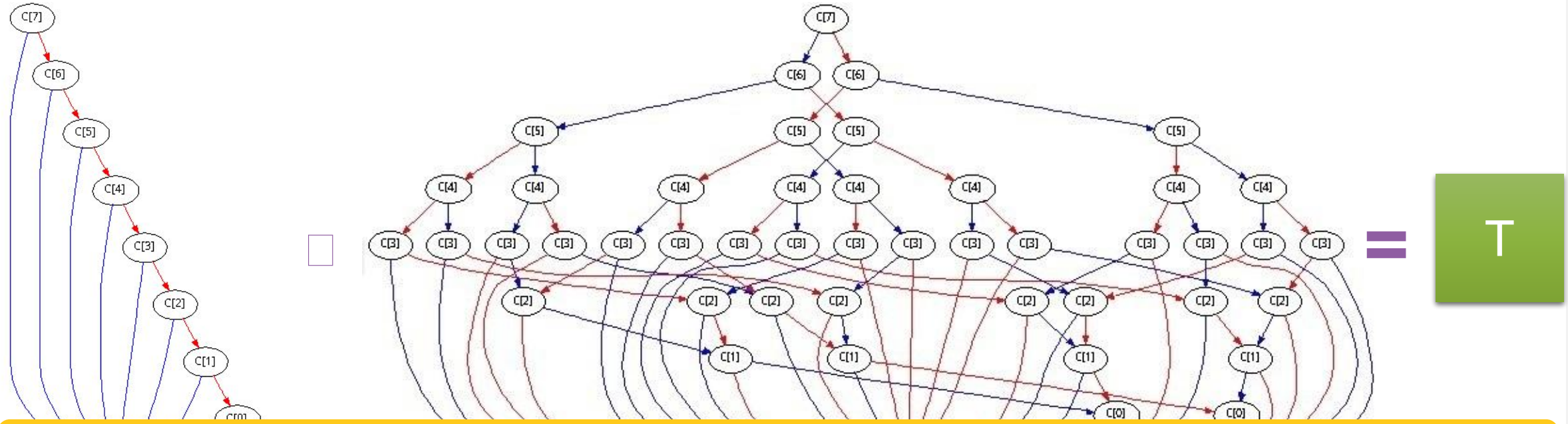
ECC Symbolic Analysis

(Countbits(C) = 0) ⇒ NE

cond

check

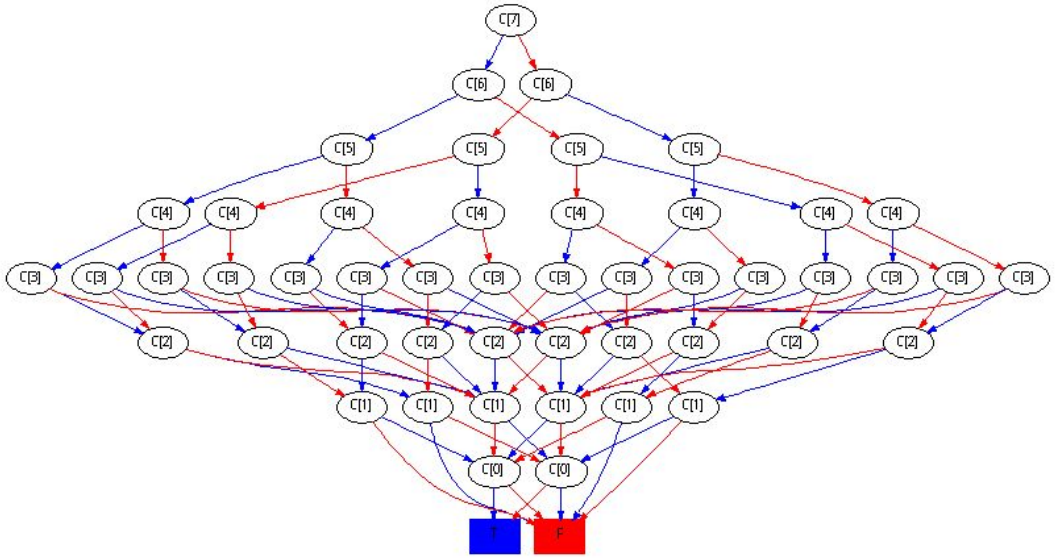
- (Countbits(C) = 0) ⇒ NE and $D_{OUT} = D_{IN}$
- 0 < Countbits(C) <= n ⇒ CE and $D_{OUT} = D_{IN}$
- (Countbits(C) = n + 1) ⇒ DUE



End-to-End ECC functionality proven - No reference model needed!

Counting & Enumerating Satisfying Paths

Example BDD for DUE (Uncorrectable Error)



Total Corruption Cases = 256

satCount(DUE) = 112

satCount((Countbits(C) = 2) AND DUE) = 28

- Access to BDDs at all circuit nodes
- Allows symbolic reasoning on BDDs
- Satisfying assignments can be counted/enumerated under various conditions
- Helps in verifying fuzzy claims like:
  - “detects ~99.999% of error patterns”

Results & Conclusion

Results

Algorithm	Data Size (in bits)	Gate Count Range	Property Convergence		
			Symbolic Simulation	EDA Tool #1	EDA Tool #2
SECDED	Up to 512	7K-270K	Yes	Yes	Yes
SECDED	4096	~1M	Yes	No	No
DECTED	256, 512	150K-500K	Yes	No	No
TECQED	512	200K-700K	Yes	No	No
Custom Reed-Solomon Algorithms	Device (32b/64b) or Column(16b) Level protection	~1M	Yes	No	No

~30 different ECC designs verified in 2 years with **same** approach, without reference models

BDD Management Techniques Used in Symbolic Simulation Proofs:
Symbolic Indexing, Parametric Substitution, Dynamic Weakening, Timed Causal Fanin Analysis, Case Splitting

Electronic Design Automation Tool #1: Model Checker; Tool #2: Commercial Datapath FV Tool

Conclusion

- ECC algorithms fit nicely in the niche of symbolic simulation.
- BDD-based symbolic simulation approach scales up to industrial ECC designs
- Provides a closed-box approach, removing dependency from a reference model
- Verification can be moved to architectural stage by replacing Writer and Reader circuits with their High-Level Models
- Enables early optimizations of algorithms by means of symbolic counting


intel