Minimization of NBTI Performance Degradation Using Internal Node Control

David R. Bild[†], Gregory E. Bok[‡], and Robert P. Dick[†]

[†]Department of EECS
University of Michigan
Ann Arbor, MI 48109, USA

[‡]Nico Trading
311 S. Wacker Drive, Suite 900
Chicago, IL 60606, USA

This work was performed while the authors were with the Department of Electrical Engineering and Computer Science, Northwestern University, Evanston IL 60208, USA.

April 21, 2009

Outline

Outline

What is NBTI

Negative Bias Temperature Instability

- Breakdown of Si–H bonds which leads to increased threshold voltage
- Reduction in transconductance and thus switching speed
- Significant for PMOS transistors under negative bias
- Partially recovers when negative bias is removed

Contributions

Internal Node Control

- Input Vector Control (IVC) – control the primary inputs
- Internal Node Control (INC) – extend control to internal nodes

Contributions

- Analyzed the potential benefit of using INC to reduce static NBTI degradation
- Developed a fast, near-optimal heuristic for the application of INC

Results

- INC – Average 26.7% improvement relative to IVC only
- INC+IVC – Average 51.3% decrease in NBTI-induced delay

Outline

Section outline

Physical Process

Reaction-Diffusion Model

- Reaction: Electric-field dependent disassociation of Si-H bonds at Si/SiO_2 interface
- Creation of interface traps, dangling bonds, oxide charges
- Diffusion: H or H_2 diffuses into the oxide

Recovery Effect

- Stress is removed
- Hydrogen diffuses back towards the interface
- Re-bonds with Si

Analytical Models

Types of Stress

- Static: Continuously applied stress (e.g., idle functional unit)
- Dynamic: Alternating stress and recovery (e.g., operating functional unit)

Models

- Cycle-based, transistor-level simulation is too slow
- Analytical models have been developed
- Overestimate degradation for static stress

Our Model

- Aggregate, long-term effect of static stress
- Assume 10% increase in delay after 10 years (Paul et. al)

Managing NBTI

Compensating Techniques

- Guardbanding
- Gate sizing
- V_{dd} tuning
- V_{th} tuning

Mitigating Techniques

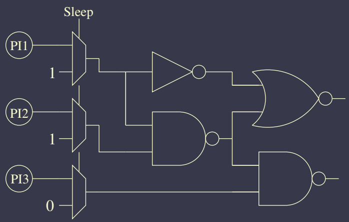
- Power gating
- Clock gating
- Input Vector Control (IVC)

Input Vector Control

Wang et al. in DATE 2006

- In sleep mode, set the primary inputs to minimize the impact on circuit delay
- Implement with MUXes, scan-chains, or modified registers
- Con: Limited effectiveness on deeper levels of logic

Example



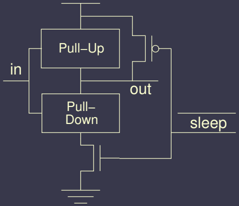
Section outline

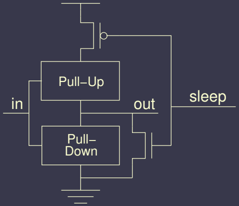
INC Implementation

Power Minimization

- Proposed by Abdollahi, Fallah, and Pedram
- Add INC to some internal gates
- Force the output value with sleep signal

INC Implementation



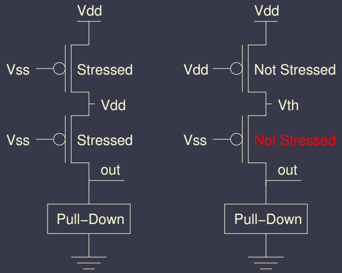


Potential for NBTI

Inverting Logic

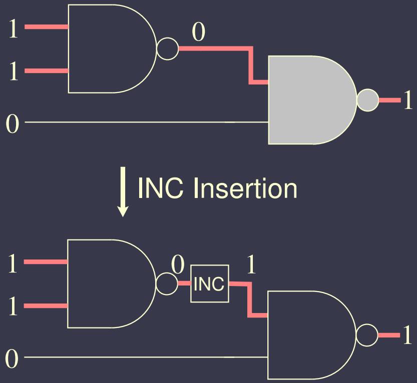
- All inputs high leads to low output
- Would have to force all outputs high to eliminate all stress
- Focus on critical path transistors instead

NOR Gate Stack



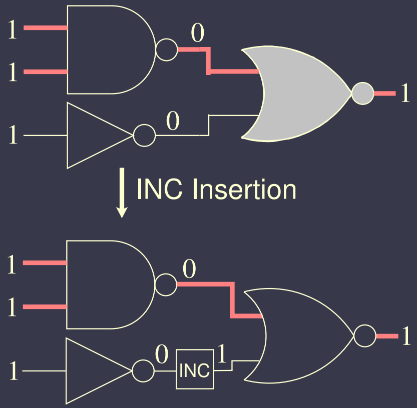
Example Insertion Scenario I

Inline



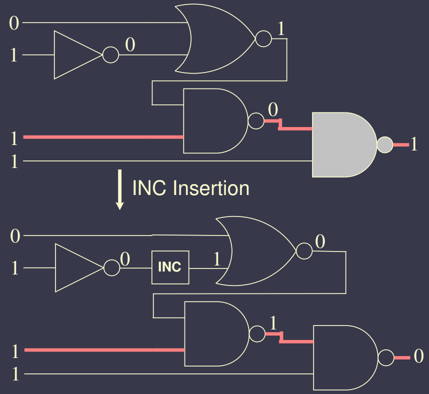
Example Insertion Scenario II

NOR Stack



Example Insertion Scenario III

Offpath



Formal Problem Definition

Given

- A combinational circuit
- For each gate, the basic delay, INC delay, and NBTI delay

Find

- Input vector
- INC insertion points

Such That

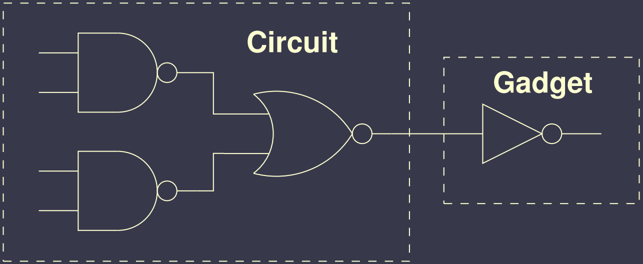
- The critical path delay (post-NBTI) is minimized

Complexity

$\mathcal{NP}$-complete

- The problem is in $\mathcal{NP}$ by polynomial-time checking
- Prove $\mathcal{NP}$-hard via reduction from circuit-SAT

Circuit-SAT Reduction



Outline

Section outline

Optimal Formulation

MILP

- MILP formulation
- Solved using Symphony open-source solver

Gate Library

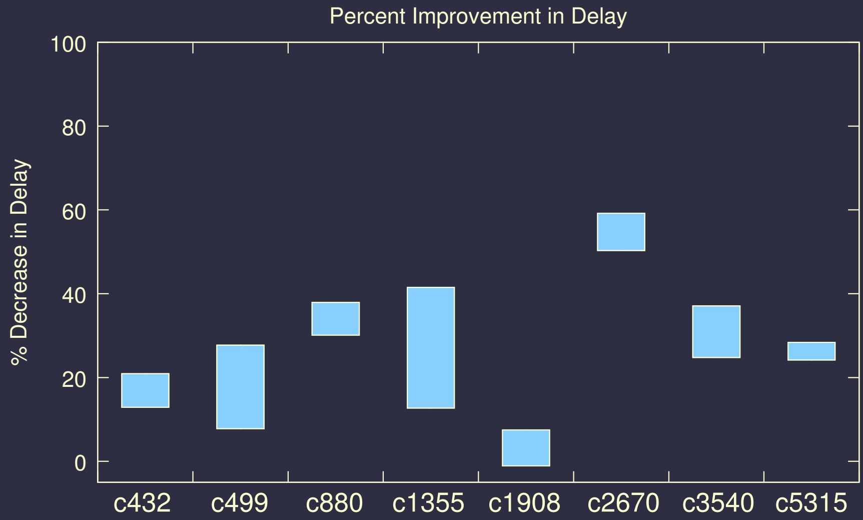
- 7 gate library in 65 nm
- Characterized timing (for INC and non-INC) using SPICE
- Verified against TSMC library

Benchmarks

- ISCAS85 benchmarks
- Synthesized using Synopsys Design Compiler

Section outline

Optimal Experimental Results I



Optimal Experimental Results II

Summary

- 26.7% average improvement relative to optimal IVC
- 51.3% average improvement relative to average IVC

Outline

Section outline

Motivation

Complexity

- Problem is $\mathcal{NP}$-hard
- For small benchmarks, can only generate upper and lower bounds on optimal

Heuristic Overview

- Linear-time
- Tree-partitioning and dynamic programming
- Based on work by Cheng, Chen, and Wong

Heuristic

```
Require: circuit $\mathcal{G}$
Require: maximum number of iterations, $N$
 1: partition circuit into trees
 2: select initial values for dangling inputs
 3: for $i = 0$ to $N$ do
 4:     for all partitions do
 5:         choose IVC and INC using dynamic programming
 6:     end for
 7:     update dangling input values
 8:     if solution is the same as previous then
 9:         break {Check for convergence}
10:     end if
11:     if oscillation is detected then
12:         repartition the circuit
13:     end if
14: end for
15: greedily remove INCs that do not affect delay
16: return  input vector and INC placements
```

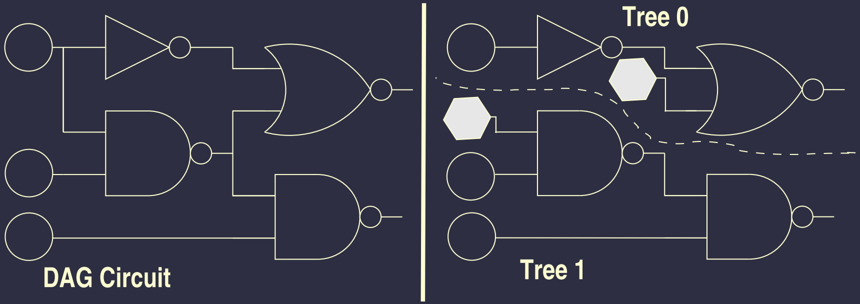
Heuristic

```
Require: circuit $\mathcal{G}$
Require: maximum number of iterations, $N$
 1: partition circuit into trees
 2: select initial values for dangling inputs
 3: for $i = 0$ to $N$ do
 4:     for all partitions do
 5:         choose IVC and INC using dynamic programming
 6:     end for
 7:     update dangling input values
 8:     if solution is the same as previous then
 9:         break {Check for convergence}
10:     end if
11:     if oscillation is detected then
12:         repartition the circuit
13:     end if
14: end for
15: greedily remove INCs that do not affect delay
16: return  input vector and INC placements
```

Heuristic

```
Require: circuit $\mathcal{G}$
Require: maximum number of iterations, $N$
 1: partition circuit into trees
 2: select initial values for dangling inputs
 3: for $i = 0$ to $N$ do
 4:     for all partitions do
 5:         choose IVC and INC using dynamic programming
 6:     end for
 7:     update dangling input values
 8:     if solution is the same as previous then
 9:         break {Check for convergence}
10:     end if
11:     if oscillation is detected then
12:         repartition the circuit
13:     end if
14: end for
15: greedily remove INCs that do not affect delay
16: return input vector and INC placements
```

Dangling Inputs



Key: ○ Primary Input ⬢ Dangling Input

Partitioning

Partitioning for Other Problems

- Proposed for other problems (e.g., leakage power minimization, tech mapping)
- Additive cost function in these cases
- Maintain large partitions

Partitioning for INC

- Maximum cost function
- Maintain critical path in a single partition
- Keep connections with lowest slack

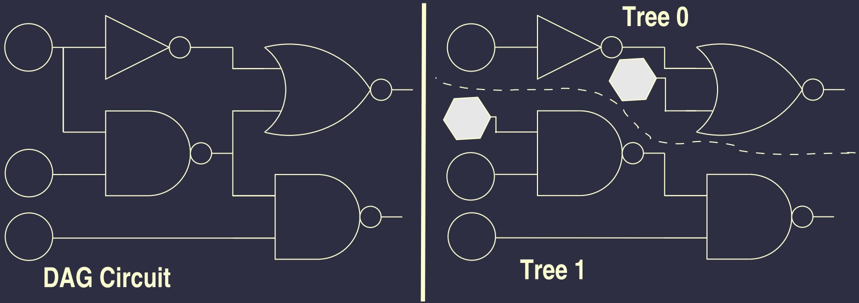
Heuristic

```
Require: circuit $\mathcal{G}$
Require: maximum number of iterations, $N$
 1: partition circuit into trees
 2: select initial values for dangling inputs
 3: for $i = 0$ to $N$ do
 4:     for all partitions do
 5:         choose IVC and INC using dynamic programming
 6:     end for
 7:     update dangling input values
 8:     if solution is the same as previous then
 9:         break {Check for convergence}
10:     end if
11:     if oscillation is detected then
12:         repartition the circuit
13:     end if
14: end for
15: greedily remove INCs that do not affect delay
16: return input vector and INC placements
```

Heuristic

```
Require: circuit $\mathcal{G}$
Require: maximum number of iterations, $N$
 1: partition circuit into trees
 2: select initial values for dangling inputs
 3: for $i = 0$ to $N$ do
 4:     for all partitions do
 5:         choose IVC and INC using dynamic programming
 6:     end for
 7:     update dangling input values
 8:     if solution is the same as previous then
 9:         break {Check for convergence}
10:     end if
11:     if oscillation is detected then
12:         repartition the circuit
13:     end if
14: end for
15: greedily remove INCs that do not affect delay
16: return input vector and INC placements
```

Dangling Inputs



Key: ◯ Primary Input ⬢ Dangling Input

Initial Assignment

Initial Assignment

- Apply dynamic programming algorithm to the full DAG circuit
- When conflicting assignments occur, choose the majority
- Choose 1 when ties occur

Heuristic

```
Require: circuit $\mathcal{G}$
Require: maximum number of iterations, $N$
 1: partition circuit into trees
 2: select initial values for dangling inputs
 3: for $i = 0$ to $N$ do
 4:     for all partitions do
 5:         choose IVC and INC using dynamic programming
 6:     end for
 7:     update dangling input values
 8:     if solution is the same as previous then
 9:         break {Check for convergence}
10:     end if
11:     if oscillation is detected then
12:         repartition the circuit
13:     end if
14: end for
15: greedily remove INCs that do not affect delay
16: return input vector and INC placements
```

Heuristic

```
Require: circuit $\mathcal{G}$
Require: maximum number of iterations, $N$
 1: partition circuit into trees
 2: select initial values for dangling inputs
 3: for $i = 0$ to $N$ do
 4:     for all partitions do
 5:         choose IVC and INC using dynamic programming
 6:     end for
 7:     update dangling input values
 8:     if solution is the same as previous then
 9:         break {Check for convergence}
10:     end if
11:     if oscillation is detected then
12:         repartition the circuit
13:     end if
14: end for
15: greedily remove INCs that do not affect delay
16: return  input vector and INC placements
```

Dynamic Programming

Forward Pass

- Compute input combination and INC state with 0 output and minimal arrival time
- Compute input combination and INC state with 1 output and minimal arrival time

Backward Pass

- Choose output value with minimal arrival time for output
- Back-propagate the values

Heuristic

```
Require: circuit $\mathcal{G}$
Require: maximum number of iterations, $N$
 1: partition circuit into trees
 2: select initial values for dangling inputs
 3: for $i = 0$ to $N$ do
 4:     for all partitions do
 5:         choose IVC and INC using dynamic programming
 6:     end for
 7:     update dangling input values
 8:     if solution is the same as previous then
 9:         break {Check for convergence}
10:     end if
11:     if oscillation is detected then
12:         repartition the circuit
13:     end if
14: end for
15: greedily remove INCs that do not affect delay
16: return input vector and INC placements
```

Heuristic

```
Require: circuit $\mathcal{G}$
Require: maximum number of iterations, $N$
 1: partition circuit into trees
 2: select initial values for dangling inputs
 3: for $i = 0$ to $N$ do
 4:     for all partitions do
 5:         choose IVC and INC using dynamic programming
 6:     end for
 7:     update dangling input values
 8:     if solution is the same as previous then
 9:         break {Check for convergence}
10:     end if
11:     if oscillation is detected then
12:         repartition the circuit
13:     end if
14: end for
15: greedily remove INCs that do not affect delay
16: return input vector and INC placements
```

Heuristic

```
Require: circuit $\mathcal{G}$
Require: maximum number of iterations, $N$
 1: partition circuit into trees
 2: select initial values for dangling inputs
 3: for $i = 0$ to $N$ do
 4:     for all partitions do
 5:         choose IVC and INC using dynamic programming
 6:     end for
 7:     update dangling input values
 8:     if solution is the same as previous then
 9:         break {Check for convergence}
10:     end if
11:     if oscillation is detected then
12:         repartition the circuit
13:     end if
14: end for
15: greedily remove INCs that do not affect delay
16: return  input vector and INC placements
```

Heuristic

```
Require: circuit $\mathcal{G}$
Require: maximum number of iterations, $N$
 1: partition circuit into trees
 2: select initial values for dangling inputs
 3: for $i = 0$ to $N$ do
 4:     for all partitions do
 5:         choose IVC and INC using dynamic programming
 6:     end for
 7:     update dangling input values
 8:     if solution is the same as previous then
 9:         break {Check for convergence}
10:     end if
11:     if oscillation is detected then
12:         repartition the circuit
13:     end if
14: end for
15: greedily remove INCs that do not affect delay
16: return input vector and INC placements
```

Heuristic

```
Require: circuit $\mathcal{G}$
Require: maximum number of iterations, $N$
 1: partition circuit into trees
 2: select initial values for dangling inputs
 3: for $i = 0$ to $N$ do
 4:    for all partitions do
 5:       choose IVC and INC using dynamic programming
 6:    end for
 7:    update dangling input values
 8:    if solution is the same as previous then
 9:       break {Check for convergence}
10:    end if
11:    if oscillation is detected then
12:       repartition the circuit
13:    end if
14: end for
15: greedily remove INCs that do not affect delay
16: return input vector and INC placements
```

Runtime

Linear in number of gates

- Partitioning – 1 topological traversal
- Dynamic programming – 2 topological traversals
- Limit the number of iterations to a small constant (e.g., 15)

Section outline

Experimental Results

Circuit	% Worse than Mid	% Worse and UB	Time (s)
c432	0.56	0.53	1.3
c499	-0.01	-0.10	6.7
c880	0.07	0.00	2.7
c1355	0.60	0.38	5.2
c1908	0.10	0.00	3.2
c2670	0.05	0.00	11.9
c3540	0.05	-0.10	27.0
c5315	0.00	0.00	41.0

Area Impacts

Circuit	Total Gates	INC Gates	Original Trans.	INC Trans.	% Increase
c432	159	11	636	22	3.5
c499	526	18	1836	36	2.0
c880	336	11	1306	22	1.7
c1355	480	12	1840	24	1.3
c1908	363	8	1322	16	1.2
c2670	592	13	2302	26	1.1
c3540	725	29	2966	58	2.0
c5315	1452	12	5650	24	0.4

Summary

- Proposed Internal Node Control (INC) for reducing static NBTI degradation on idle functional units
- Showed that INC leads to an average 26.7% reduction in delay relative to IVC alone
- Proposed a linear-time heuristic that generates near optimal results quickly

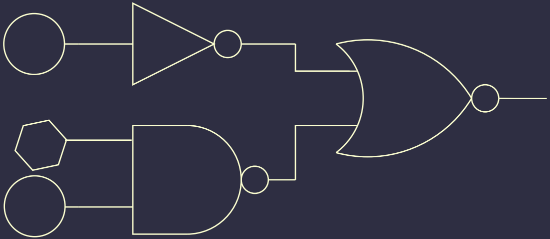
Acknowledgments

This work was supported by the Semiconductor Research Corporation under award 2007-HJ-1593 and by the National Science Foundation under awards CCF-0702761 and CNS-0347941.

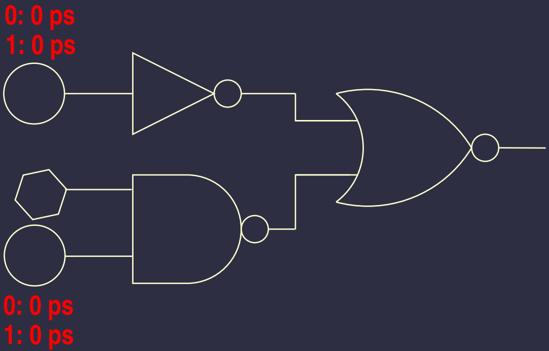
Questions

Thank You

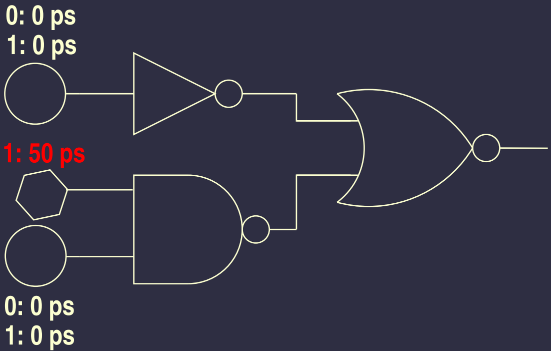
Dynamic Programming Example



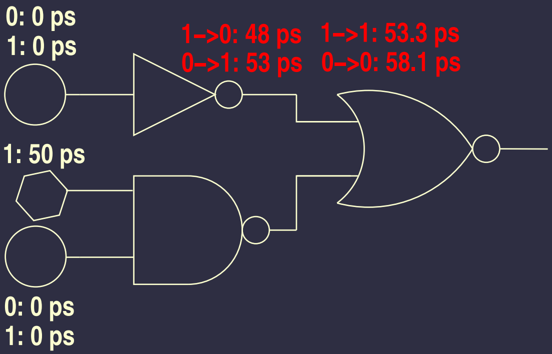
Dynamic Programming Example



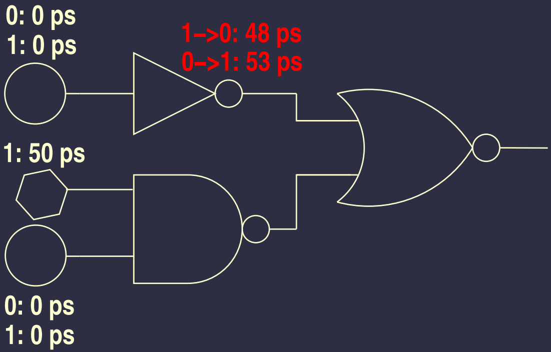
Dynamic Programming Example



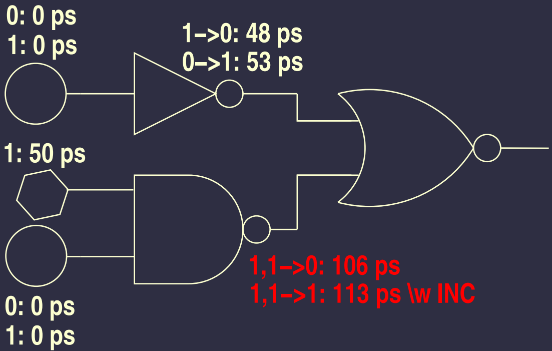
Dynamic Programming Example



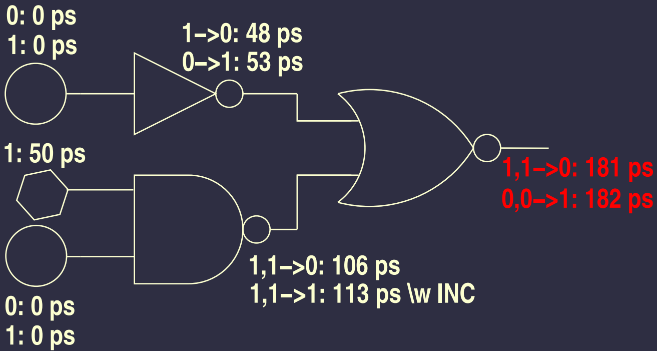
Dynamic Programming Example



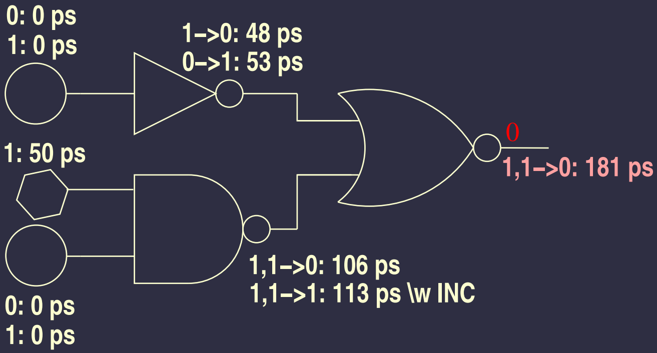
Dynamic Programming Example



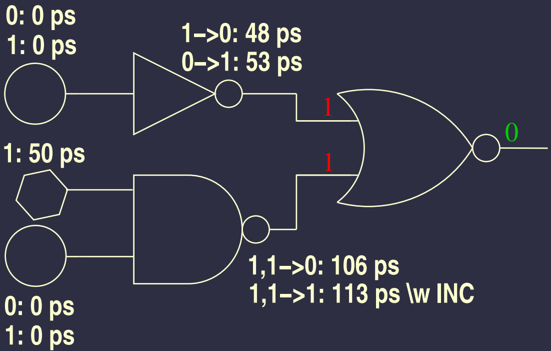
Dynamic Programming Example



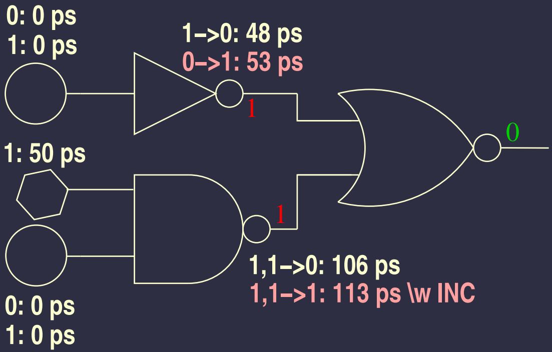
Dynamic Programming Example



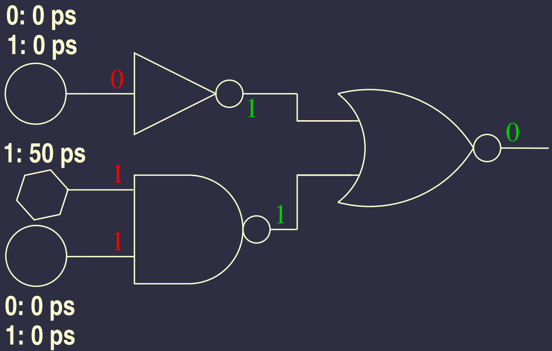
Dynamic Programming Example



Dynamic Programming Example



Dynamic Programming Example



Dynamic Programming Example