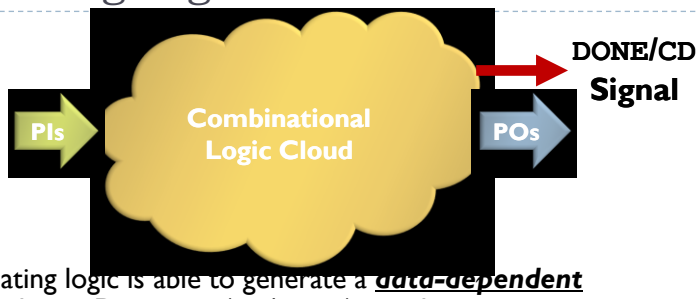
CE653 – Asynchronous Circuit Design

Instructor: C. Sotiriou

http://inf-server.inf.uth.gr/courses/CE653/

Contents

Indicating Logic



- Indicating logic is able to generate a ***data-dependent*** Completion Detection (CD/DONE) signal
  - Indicates that **valid data** have arrived at the C.L. cloud's outputs
- What was there before valid data arrived?
  - difficult separating successive data values ($Valid_n$, $Valid_{n+1}$, …)
- Intermediate value between valid data used (NULL/Spacer)
  - Indicating circuit operation is two-phase: NULL, DATA, NULL, DATA, …

Indicating Logic - Encodings

- Indicating circuit operation is two-phase:
  - NULL, DATA, NULL, DATA, …
- Boolean Logic encodes two values:
  - 0 or 1, T or F
- We need DATA and NULL, *i.e.* three values
  - 0 or 1 or NULL, T or F or NULL
- Simplest ternary logic is **`dual-rail`**
  - ***Need two wires per Boolean signal***
- Other Encodings Possible
  - **`1-of-n`** or **`m-of-n`**

Actual Value: `x.t`, `x.f`	Logical/Boolean Value
00	NULL
01	0 (False)
10	1 (True)
11	Unused

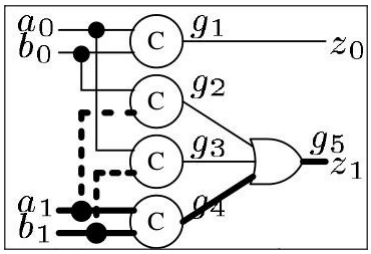
Indicating Logic – Strong vs. Weak Indication

- Dual-rail encoding allows us to detect `DATA` arrival at
  - Inputs or Outputs
- Definition[**Strongly/Weakly Indicating Logic**]
  - A circuit is *strongly-indicating* iff it waits for all of its inputs to arrive before it computes and produces valid outputs, as well as for all of its inputs to become NULL before it produces NULL outputs

Indicating Logic – Orphan Wires/Gates

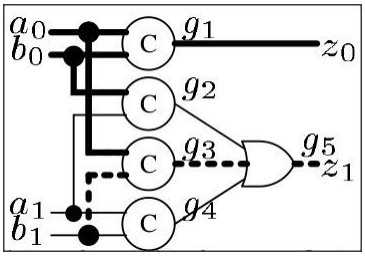
- Orphans represent *unacknowledged* circuit nodes at the POs
- Definition[**Wire Orphan**]
  - An *orphan wire*, or *wire orphan* is an indicating circuit wire, a signal transition of which, triggered by a given and valid input transition, is **NOT** acknowledged by a respective signal transition **on any PO**
- Definition[**Gate Orphan**]
  - An orphan gate, or gate orphan is an indicating circuit gate, a signal transition through which, triggered by a given and valid input transition, is **NOT** acknowledged by a respective signal transition **on any PO**

Indicating Logic – Orphan Wires/Gates



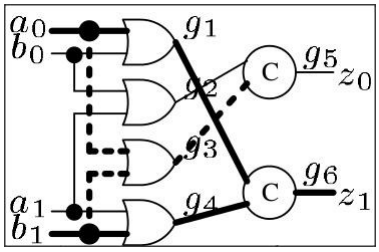
- Thick lines
  - data evaluation
- Dotted lines
  - wire orphans which do not propagate, and are unacknowledged, but must be cleared before new data

Indicating Logic – Orphan Wires/Gates



- Orphan on lower input wire of `g3` causes erroneous output on `z1`

Indicating Logic – Orphan Wires/Gates

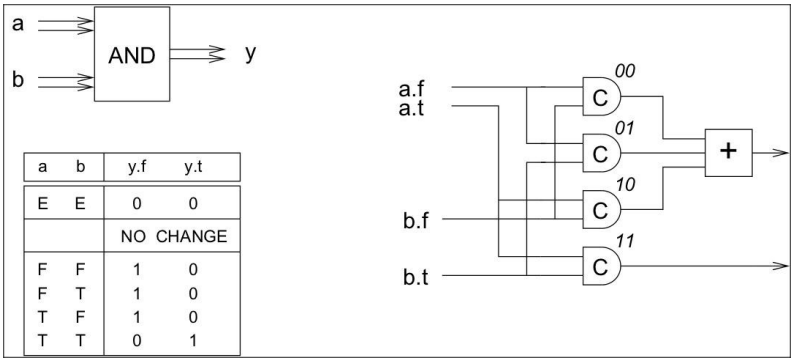


- In this example, orphans propagate through gates
- For proper reset dotted lines MUST return to zero
- But this cannot be checked by inspection of the circuit's outputs!

DIMS – Delay Insensitive Minterm Synthesis

DIMS

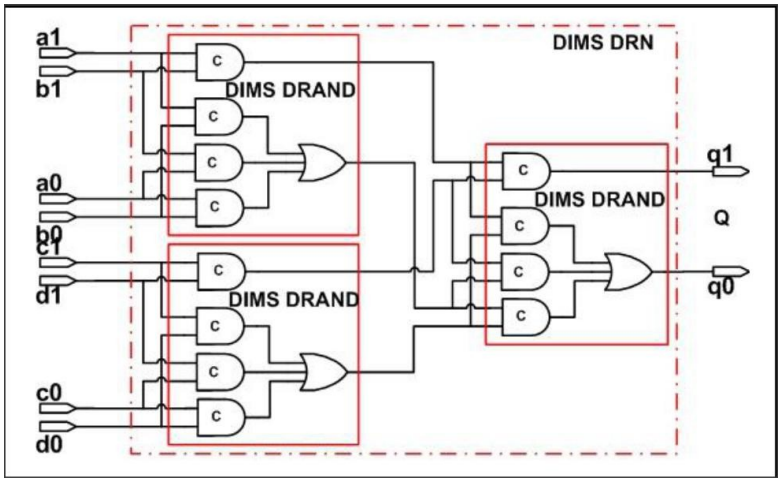
- DIMS is the simplest form of DR logic
  - Each signal is instantiated in `.t` and `.f` rails
  - Truth-table Minterms directly translated to **2-level** DR logic
    - **1st DR circuit level is C-Elements, 2nd level is OR gates**



a	b	y.f	y.t
E	E	0	0
		NO CHANGE	
F	F	1	0
F	T	1	0
T	F	1	0
T	T	0	1

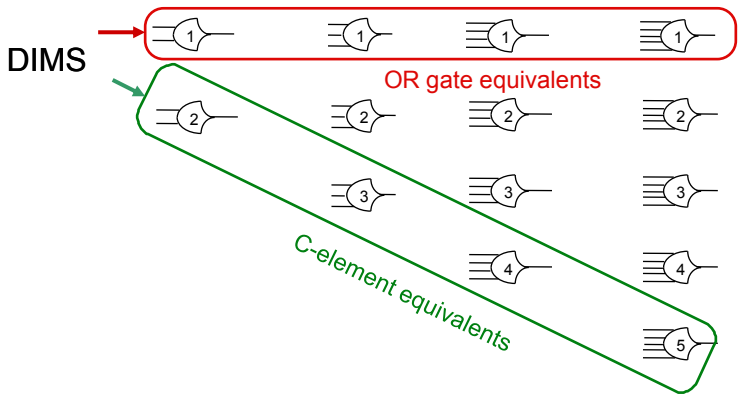
DIMS (Delay Insensitive Minterm Synthesis)

- Four-input NAND Example in DIMS



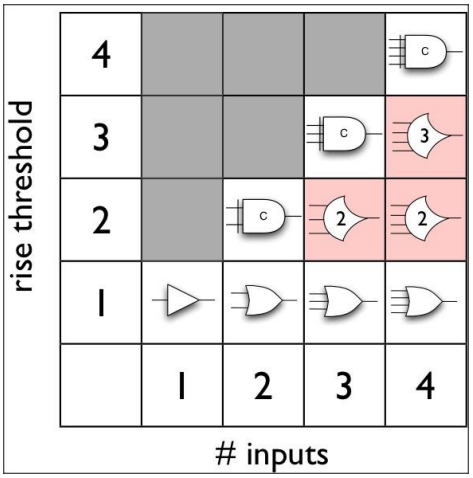
NCL – Null Convention Logic

Threshold Gates – DIMS Generalisation



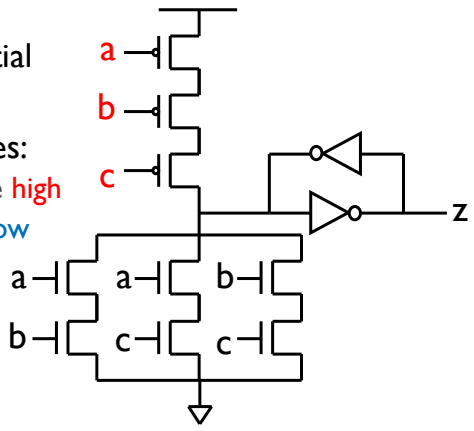
Threshold Gates – DIMS Generalisation

- Threshold gates
  - generalisation of C-elements
- *Threshold Gate Output Rises* when the **number of threshold inputs *rises***
- *Threshold Gate Output Falls* when **all inputs *fall***



Threshold Gates – 2 of 3 with hysteresis

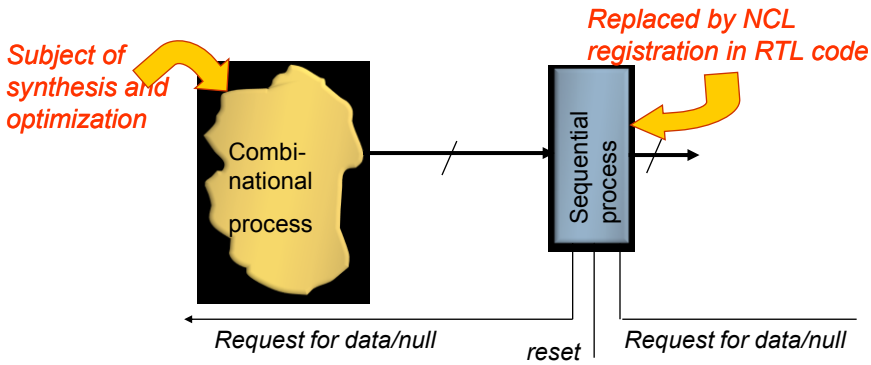
- Hysteresis means Sequential
- Threshold 23 gate switches:
  - high when 2 of 3 inputs are high
  - low when all 3 inputs are low



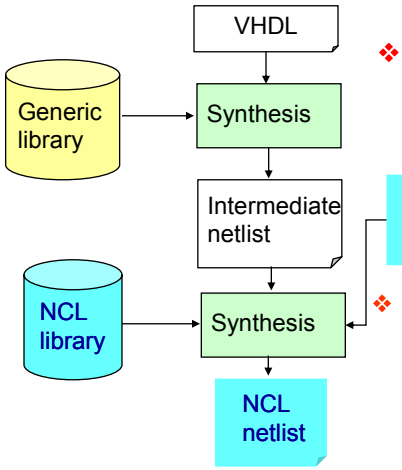
- Boolean Function Format:
  - `z = ab + ac + bc + z(a + b + c)`

NCL Flow 1 – Separate C.L., Registers

- Combinational Logic and Registers are separated



NCL Commercial Flow – Synopsys DC Based



- Step 1. Translate HDL into "synchronous" netlist
- Step 2. Convert intermediate netlist into NCL netlist

NCL Optimisation with Synopsys DC

- Dual-rail expansion
- Two phases (set and reset) are separated
- Set phase ensures circuit functionality
- Reset phase is implied
- Optimizations are applied to the set phase

NCL Gate Images – Output Rise (Set) Equivalent

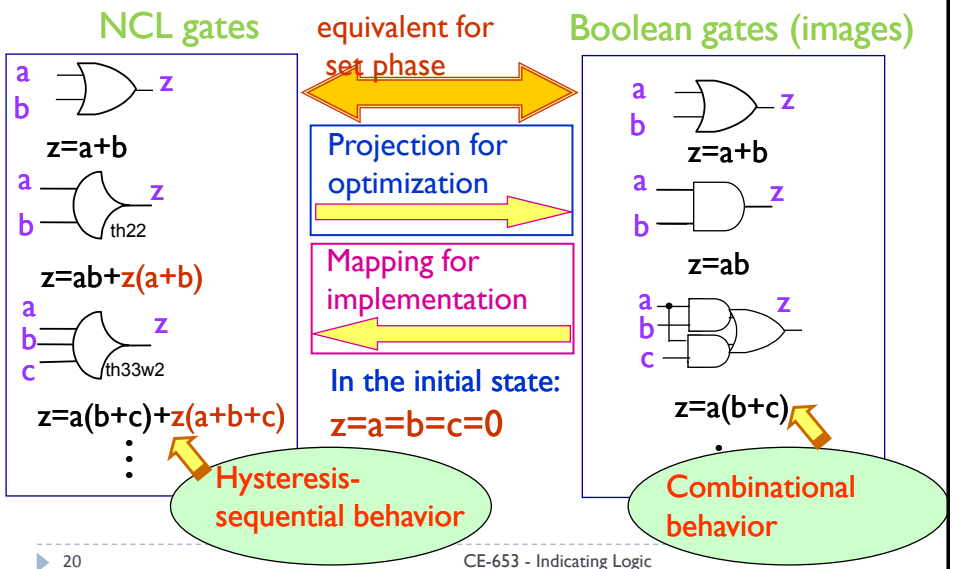
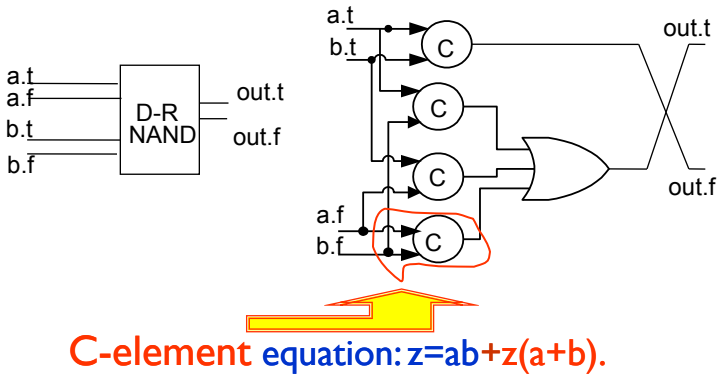


Image of D-R NAND Gate

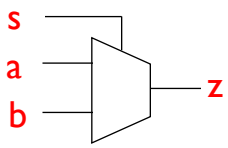


- No Difference to DIMS implementation of standard-cell gates

NCL Flow Detailed Example – Step 1

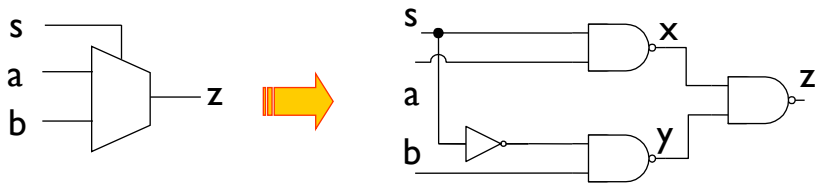
- Conventional RTL Description
  - Multiplexer

```
entity test
   input a,b,s : ncl_logic;
   output z    : ncl_logic;
architecture
process (a, b, s) is begin
   if s = '1' then
      z <= a;
   else
      z <= b;
   end if;
end process;
```



NCL Flow Detailed Example – Step 2

- Conventional RTL to Boolean Gates Synthesis

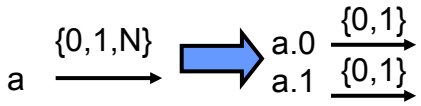


Two input NAND gates

NCL Flow Detailed Example – Step 3

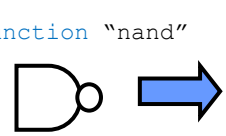
- Define new type for signal logic – `dual_rail_logic`

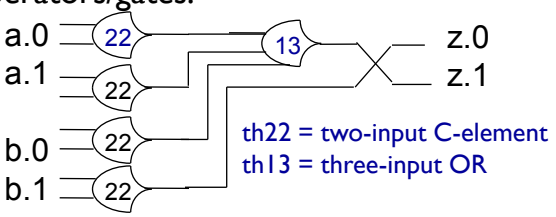
```
type dual_rail_logic is record
      rail1 : std_logic ;
      rail0 : std_logic ;
end record;
```



- Overload common operators/gates:

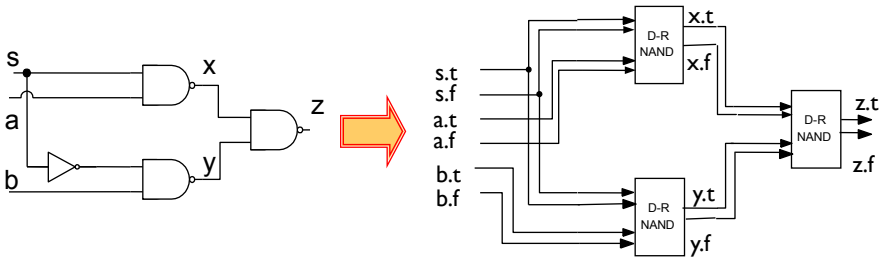
`function "nand"`





`function "not"`

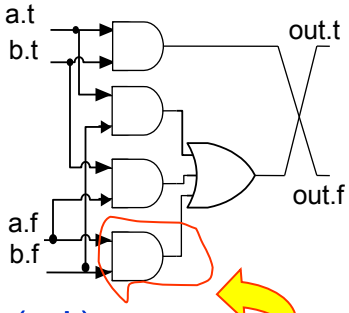
NCL Flow Detailed Example – Step 4



- Naive semi-static DIMS implementation – 114 transistors - *may be reduced to 63 transistors by merging C-elements with OR-gates* - versus 14 for a synchronous circuit

NCL Flow Detailed Example – Set Phase

- During the ***set*** phase C-elements in D-R gates ***behave like AND gates***

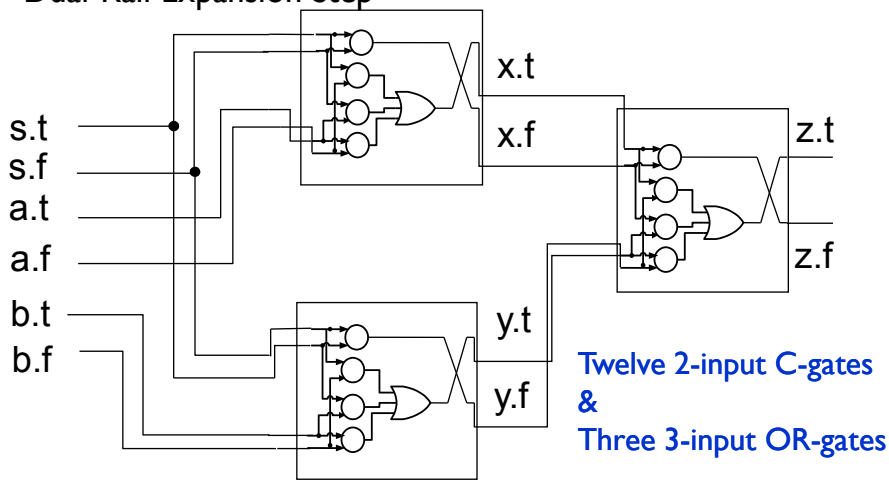


C-element equation: z=ab+z(a+b), initially z=a=b=0

In a set phase it behaves like an AND gate z=ab

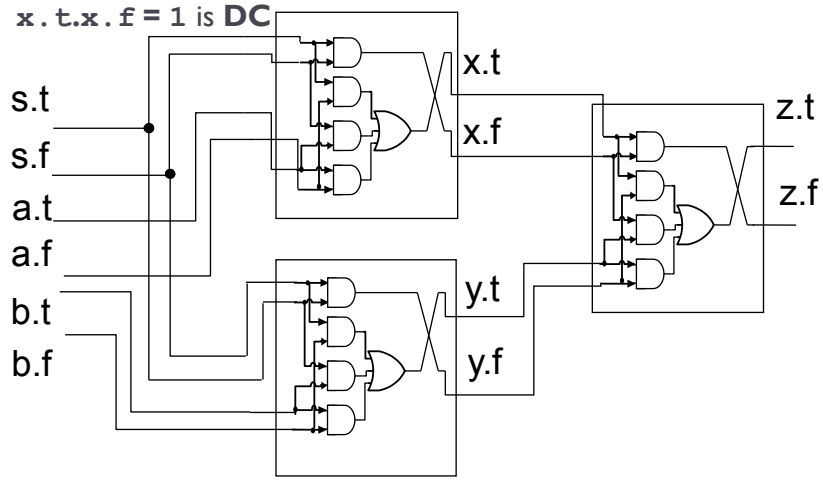
NCL Flow Detailed Example – Step 5

- Dual-Rail Expansion Step



Set Phase Image Circuit after DR Expansion

- This circuit contains DCs
  - `x.t.x.f = 1` is **DC**



NCL Flow Detailed Example – Step 6

- Technology independent optimization (DCs)
- Technology-mapping of image gates to NCL library gates

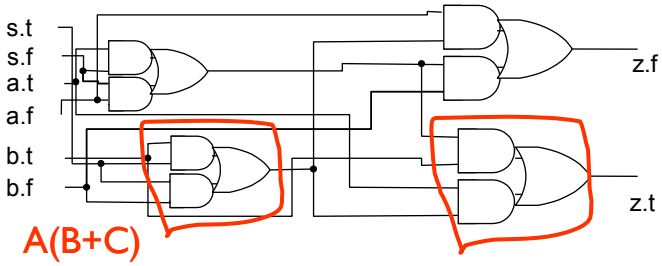
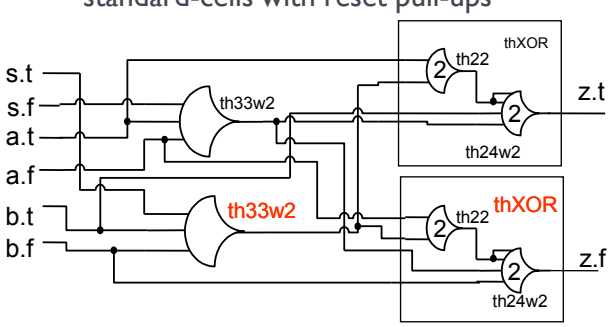


image of th33w2

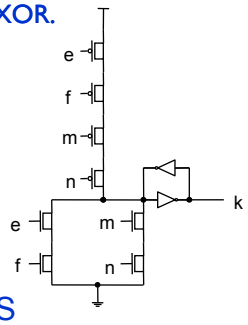
image of thXOR

NCL Flow Detailed Example – Step 7

- Final NCL circuit
  - Image gates (Set phase) are replaced by sequential NCL standard-cells with reset pull-ups

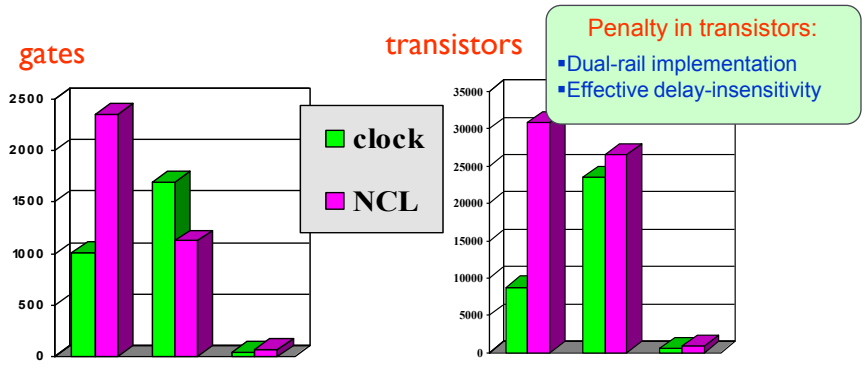


Semi-static CMOS implementation of thXOR.



44 transistors - 30% better than optimized DIMS

NCL Flow Results



- Typically, actual area overhead is >2.5X

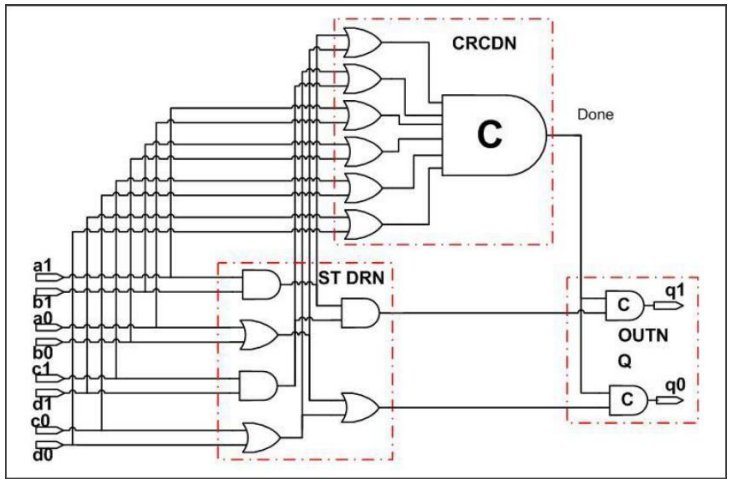
NCLX – NCL with eXplicit completion

NCLX – Explicit Completion

- Aims to reduce huge area overhead of NCL
- Strongly-Indicating Logic (alike DIMS/NCL)
- Key Idea
  - **Separate Functional Part** (Set Functions for DR Outputs) **and Delay-Insensitive Part** (Resetting and Orphans)
- Four Boolean Networks
  - Dual-Rail Functional Part (DR Inputs → DR Outputs)
  - Input Completion Part for Strong-Indication
    - Also referred to as . `go` signal
  - Intermediate Node Completion Part for Orphan Elimination
  - Output Completion Part for Strong-Indication
- Input Completion and Local Completion are merged

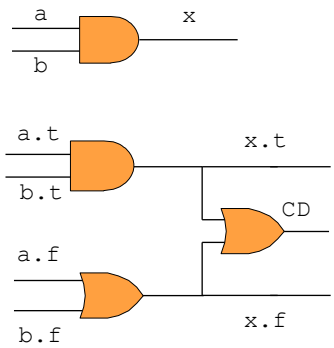
NCLX

- Four-input NAND Example in NCLX



Dual-Rail Network Conversion - MBN

- Weakly-Indicating
  - No input, local node completion
  - Timing assumptions required for orphan nodes/gates



Original BN:

$x = a\,b$

$[x' = a' + b']$

DR Equivalent BN:

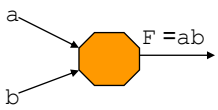
$x.t = a.t\,b.t$

$x.f = a.f + b.f$

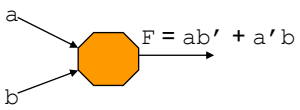
$CD = x.t + x.f$

Monotonic Boolean Network

- Boolean Network consists of
  - Monotonic Nodes
    - Must be **Unate**
    - Unate = node function contains each variable in either non-complemented or complemented form

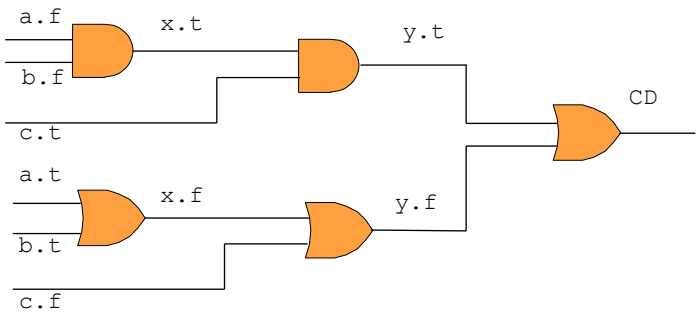


Monotonic Node

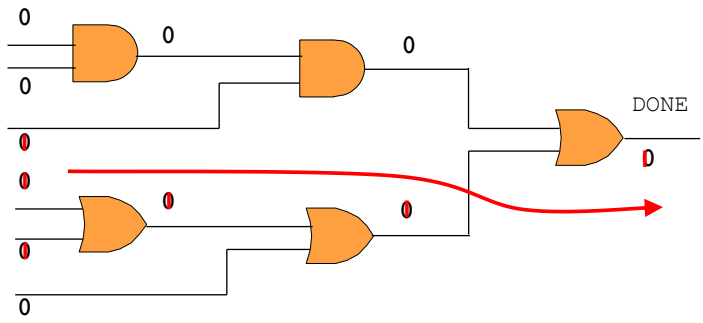


Non-Monotonic Node

Two Phase DR Network Elastic Operation

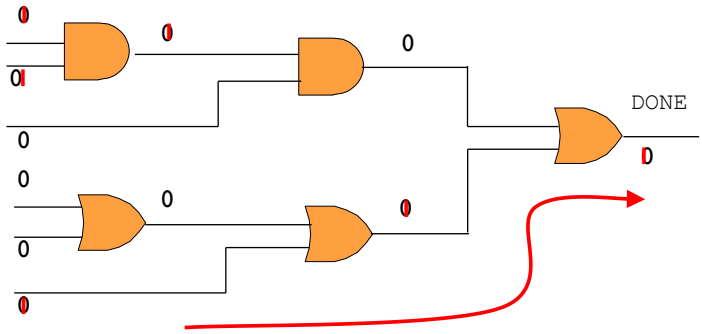


Two Phase DR Network Elastic Operation



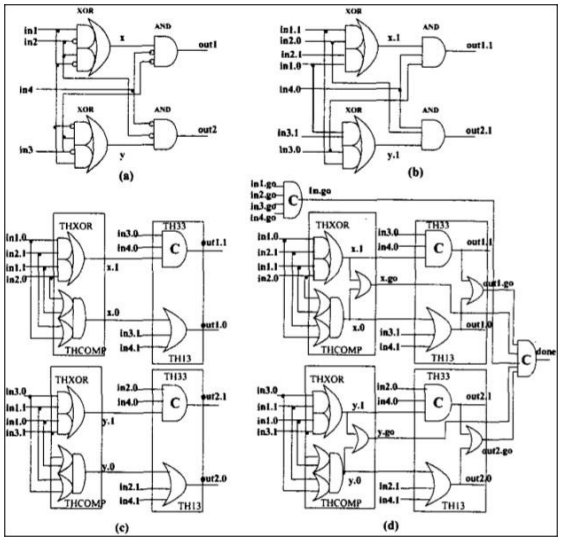
Critical path depth: 3 Logic Levels (LL)

Two Phase DR Network Elastic Operation



Critical path depth: 2 Logic Levels (LL)

Complex NCLX Example



NCLX Results

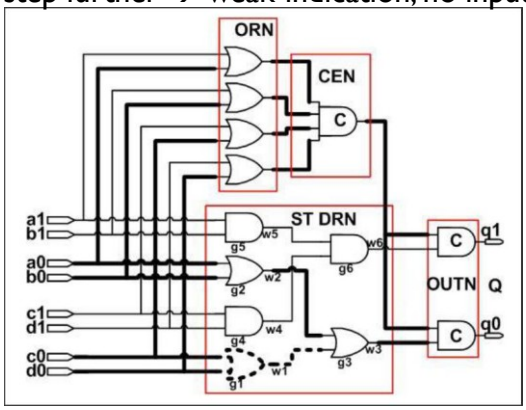
- Good improvement over NCL
- Difficult to reduce area further without sacrificing delay-insensitivity

Table 1. Area Comparison

Circuit	NCL_D	NCL_X	%
AND16	480	186	-61%
AND4	96	30	-69%
IF_THEN_ELSE	72	38	-47%
HA	72	56	-22%
FA	176	118	-33%
CLIPPER	448	237	-47%
MUX_DECODER	456	352	-23%
SET_CNT	490	456	-7%
USHIFT	1264	762	-40%
NCL_ADDRCONV	1356	1045	-23%
SERIAL_CRC	2010	1936	-4%
SYNC_STATE	2402	1934	-19%
BIT_CNT	2779	2432	-12%
FSM_DATAPATH	10260	9290	-9%
NCL_X2	15338	13852	-10%
NCL_X1	18606	12724	-32%
VITERBI DECODER	45198	38130	-16%
Average	-	-	**-28%**

Timing Assumptions for DR Logic

- Strongly-Indicating DR Circuit – detect arrival of all inputs
- **Timing Assumption about Orphan nodes' Reset**
- Can go a step further → weak-indication, no input detection…

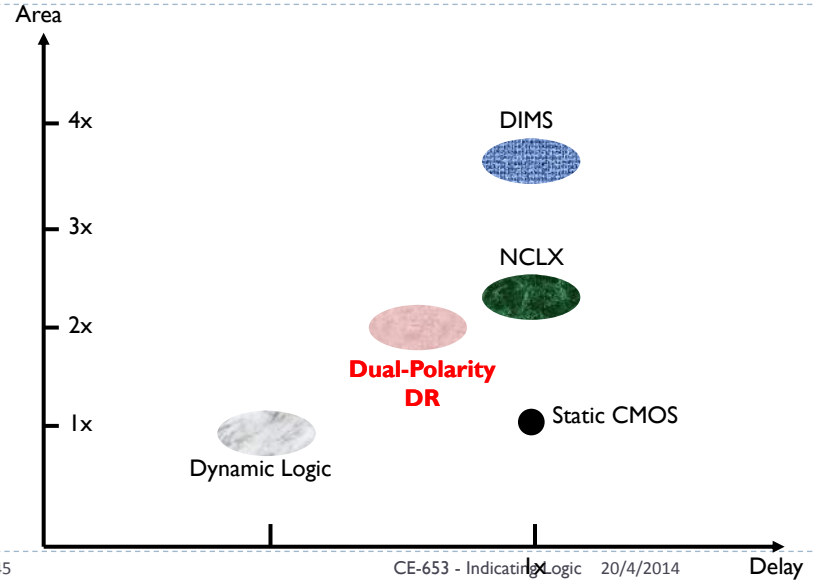


Dual-Polarity/Phase DR Logic

Dual-Polarity DR Logic Methodology

- All methodologies so far only support **positive** polarity gates (positive unate)
  - AND/OR, etc.
- Dual-Polarity DR Logic supports both **Positive** and **Negative** Polarity Gates
  - e.g. NAND, NOR, etc.
- CMOS Negative gates are ***faster*** than positive gates
  - Positive gates are Negative gates + inverter
- ***General methodology for Boolean Network transformation to a Monotonic Boolean Network***
  - Logic Synthesis level

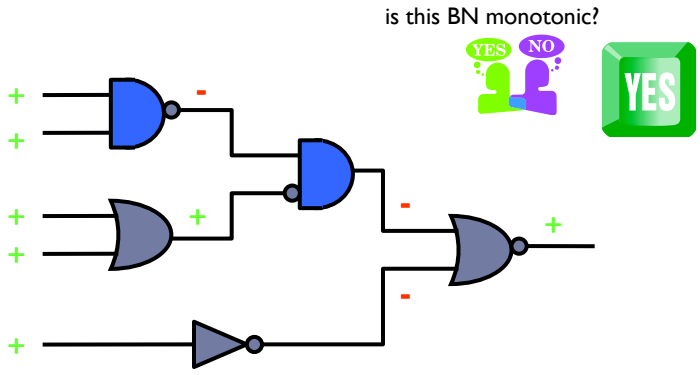
Dual-Polarity DR Logic Methodology



MBNs (Monotonic Boolean Networks)

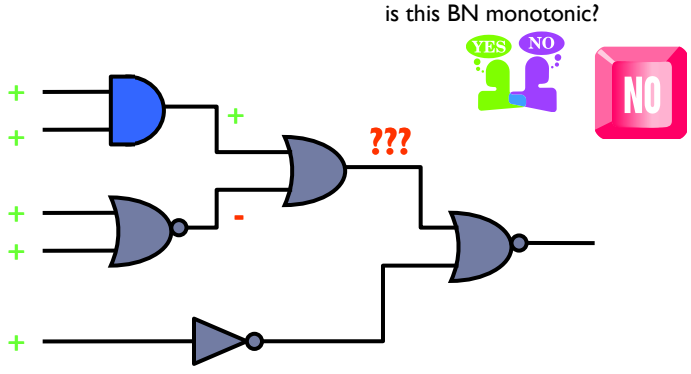
- We can allow for BN nodes to be phased as negative or positive
- Definition[**Increasing/Decreasing Node**]
  - BN node *f* is *increasing (decreasing)* in positive (negative) variable x_i → x_i: **0**→**1** (**1**→**0**), *f* cannot change **1**→**0** (**0**→**1**)
- Definition[**Unate Function**]
  - BN function *f unate* in x_i, iff *f* is increasing or decreasing in x_i
- Definition[**Positive/Negative Nodes**]
  - Node n_i with local function f_i is *+ve (-ve)*
  - x_i is *+ve (-ve)* and f_i is ***increasing*** in x_i, OR x_i is *-ve (+ve)* and f_i is ***decreasing*** in x_i.
- Definition [**Monotonic Node**]
  - A node *is monotonic* if it is *either +ve or –ve*.
- Definition [**Monotonic Boolean Network**]
  - A BN is MBN if all its nodes are monotonic
- **MBN is *hazard-free* under *monotonic input transitions***

Monotonic Boolean Networks



- Assign polarity to every node to check monotonicity.

Monotonic Boolean Networks



MBN Transformations

- Key Questions:
  - given a BN, how do we transform it to an MBN?
  - given an MBN, which transformations and optimizations can we apply to reduce to another MBN?
- Answers:
  - Provide two transformation procedures, *based on the dual-rail code*, for generating an MBN from a generic BN:
    - Technology-Independent (TI) Conversion
    - Technology-Mapped (TM) Conversion
  - Provide a set of hazard-non-increasing transformations on MBNs.

Technology-Independent DR Conversion

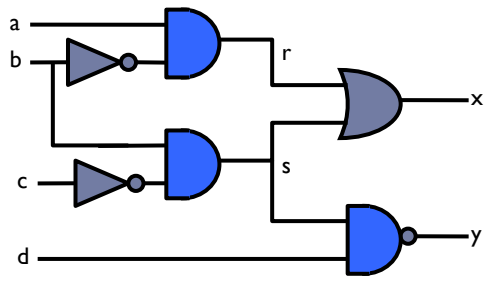
- For each PI x, create x^t and x^f representing the true and false evaluations of x.
- For each node implementing $y_i = f_i(x_1, \ldots, x_n)$, create two nodes:
  - $y^t{}_i$ = **DR**$(f_i(x_1, \ldots, x_n))$ and $y^f{}_i$ = **DR**$(f_i'(x_1, \ldots, x_n))$
  - $y = x'$, special case, $y^t = x^f$, $y^f = x^t$
- Define DR recursively (based on Shannon Expansion Th.):
  - **DR**$(0) = 0$, **DR**$(1) = 1$
  - **DR**$(x.f_x + x'.f_{x'}) = x^t$.**DR**$(f_x) + x^f$.**DR**$(f_{x'})$
- Theorem[**DR Conversion**]
  - Given function $y = f(x_1, \ldots, x_n)$, under the assumption that $x^i = x^t{}_i = x^f{}_i'$ it holds that $y = y^t = y^{f\prime}$
  - Proof: by induction on function **DR**

Technology-Independent DR Conversion

- $y = a'b + b(c + d')$
  - would be converted into:
- $y^t = \mathbf{DR}(a'b + b(c + d')) = a^t b^f + b^t(c^t + d^f)$
- $y^f = \mathbf{DR}(a'b + b(c + d'))' = (a^f + b^t)(b^f + c^f d^t)$
- How do I convert?
  - Use Shannon's Expansion Theorem
  - assume that $x_i = x^t_i = x^f_i{}'$, for all nodes/POs x in circuit
  - Use the SIS **dr package**!

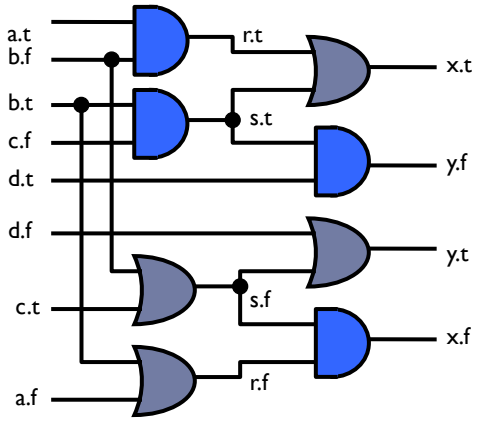
Technology-Independent Conversion - Example

- Original BN:
  - $y = a'b + b(c + d')$



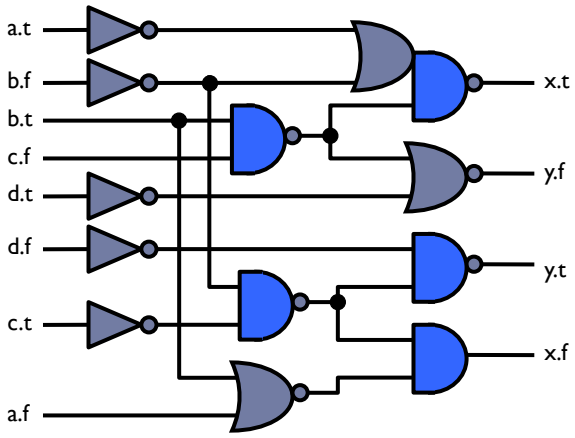
Technology-Independent Conversion - Example

- Dual-Rail Conversion:
  - $y^t = a^t b^f + b^t(c^t + d^f)$
  - $y^f = (a^f + b^t)(b^f + c^f d^t)$



Technology-Independent Conversion - Example

- Technology Mapping to a library

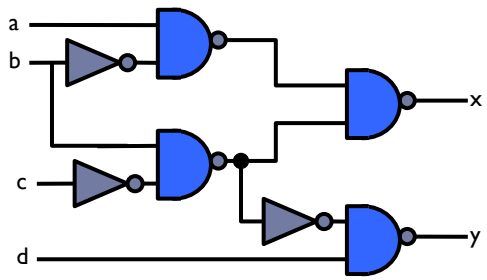


Technology-Mapped Conversion

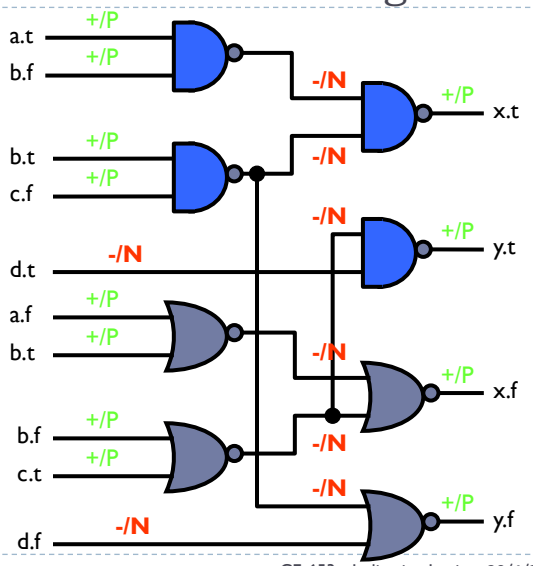
- For each gate, producing signal y^t_i, from signals $y^t_j, \ldots, y^t_k$, *add a **dual** gate*, based on De Morgan's law.
- Label each node as **+ve** or **–ve**, starting from the PO's, according to gate polarities
- In case of multiple paths from POs to the node, consider the longest
- For each inconsistently labeled gate input or PI, which is, invert and connect to its dual

Technology-Mapped Conversion - Example

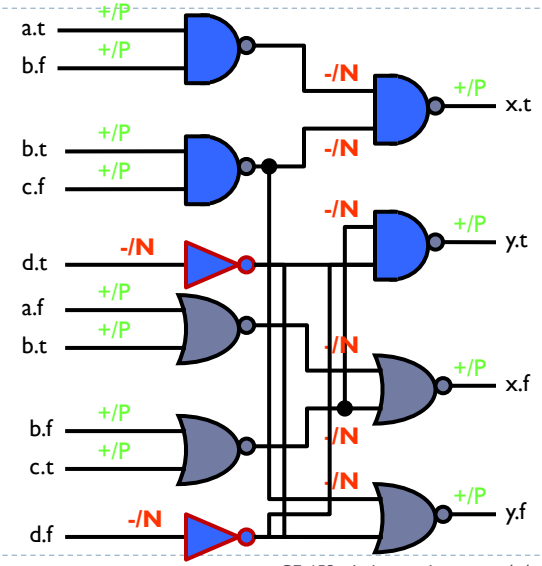
- Original technology-mapped circuit:



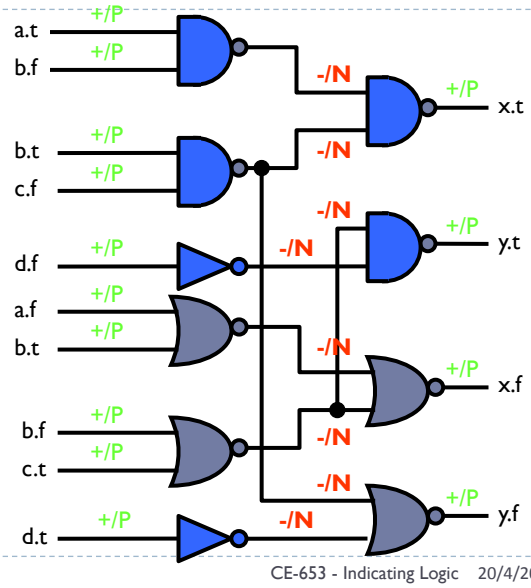
Technology-Mapped Conversion – Example – Phase Labeling



Technology-Mapped Conversion – Example – Phase Correction



Technology-Mapped Conversion – Example – Final Result



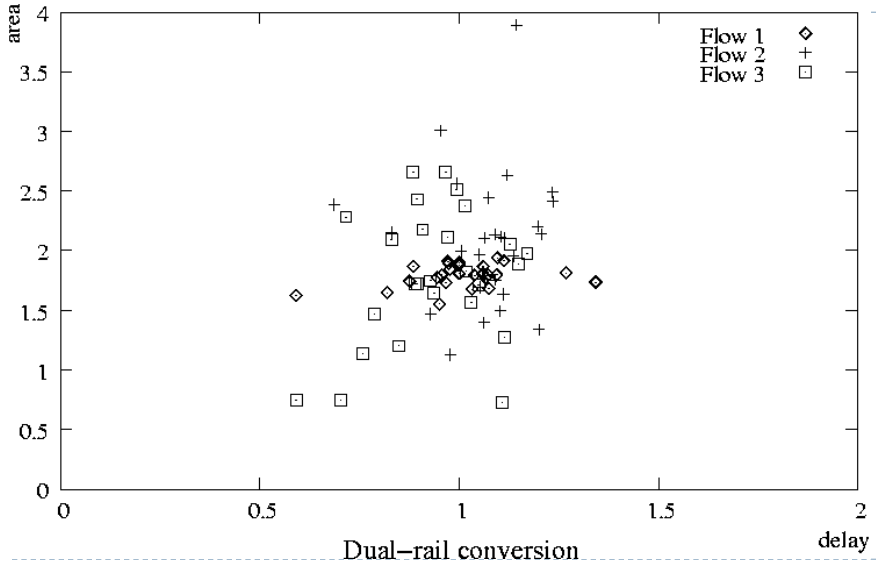
Hazard-non Increasing Transformations

- Set of transformations that preserve MBN:
  - De Morgan's laws.
  - Dual global and global flow.
  - Tree decomposition.
  - Gate replication.
  - Collapsing.
  - Kernel-factoring.
  - Cube-factoring.

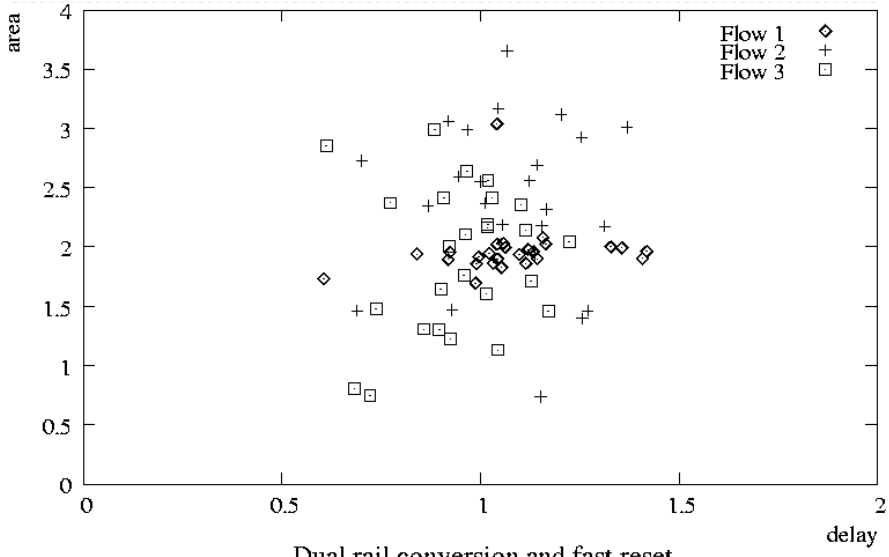
Fast Reset

- The MBN approach is NOT delay-insensitive
- Set function performs Completion-Detection
- Reset is ***timed***
- Note that typically MBN requires reset at every cycle
  - This is an issue for speed
- **Fast Reset Approach**
  - *Slice DR Logic Circuit with multiple Reset Levels*
  - Reset each level
  - Use a delay element to wait for Reset

Dual-Phase DR Results – Scatter 1

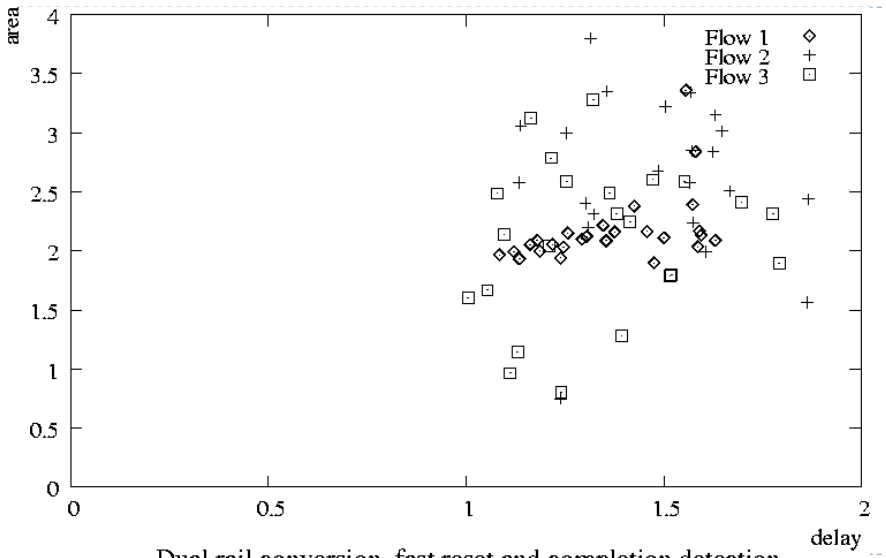


Dual-Phase DR Results – Scatter 2



Dual rail conversion and fast reset

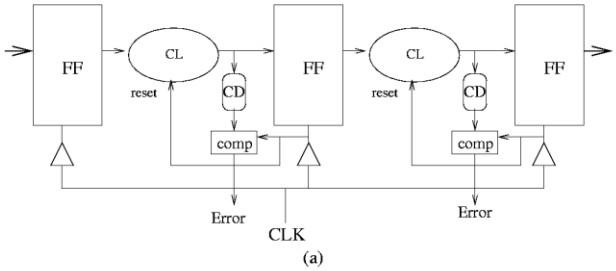
Dual-Phase DR Results – Scatter 3



Dual rail conversion, fast reset and completion detection

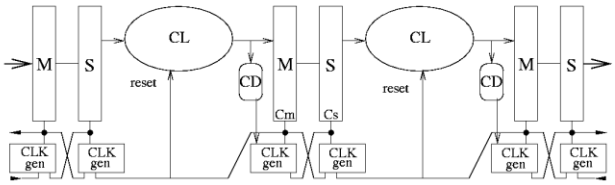
Dual-Rail MBN Use Cases

- Synchronous Environment
  - Check against a clock signal



(a)

- Asynchronous Environment
  - Exploit data-dependent latencies



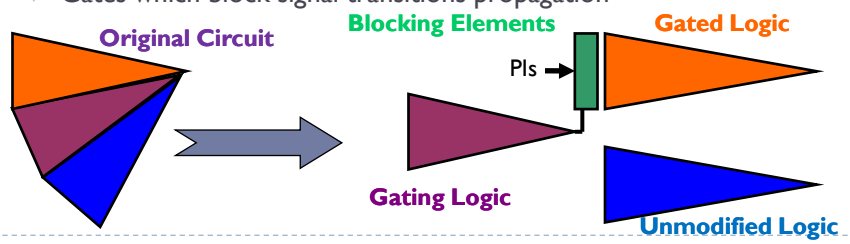
(b)

Advanced DR Methodologies

Gated DR Circuits

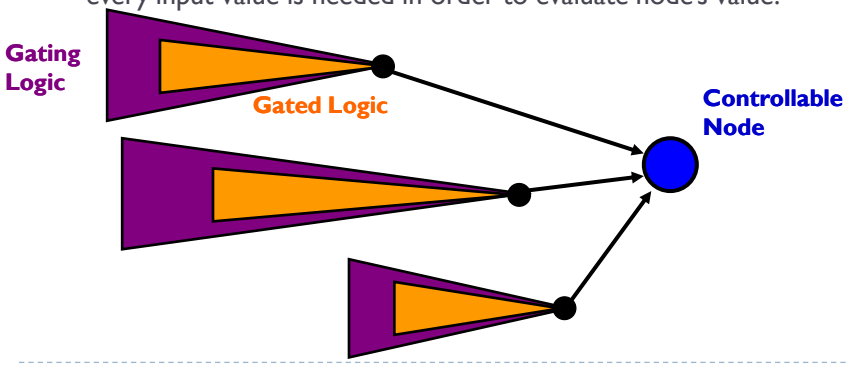
Gated DR Circuits

- GDR motivation
  - High degree of power overhead of monotonic DR circuit (~x9)
- Gated Logic
  - Logic subcircuit prevented from switching
- Gating Logic
  - Logic controlling (enabling/disabling) when the Gated Logic is prevented from switching
- Blocking Elements
  - Gates which block signal transitions propagation

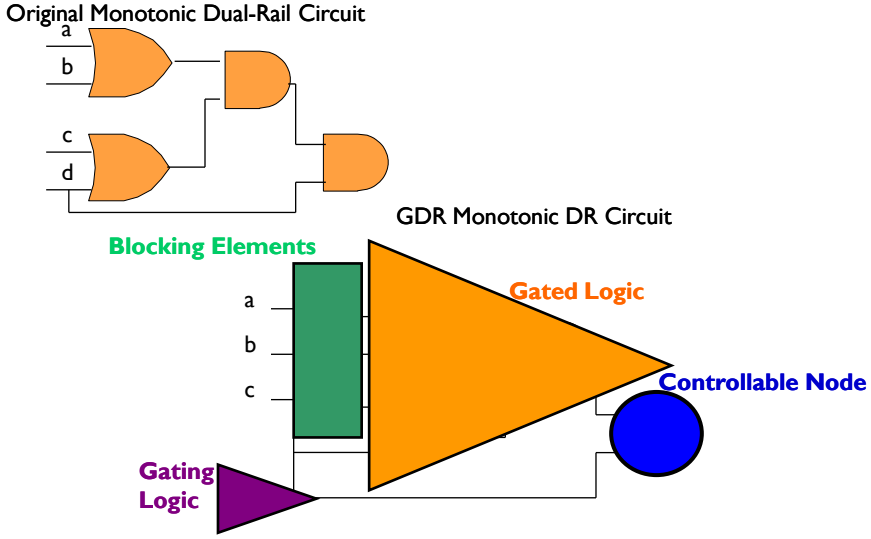


Gated DR Circuits – Basics

- A Circuit node is controllable if a subset of its inputs is able to determine node's value
  - A node implementing the AND logic function is controllable, as the evaluation of any of its inputs to 0 is able to determine node's value.
  - A node implementing the XOR logic function is not controllable, as every input value is needed in order to evaluate node's value.



GDR Example

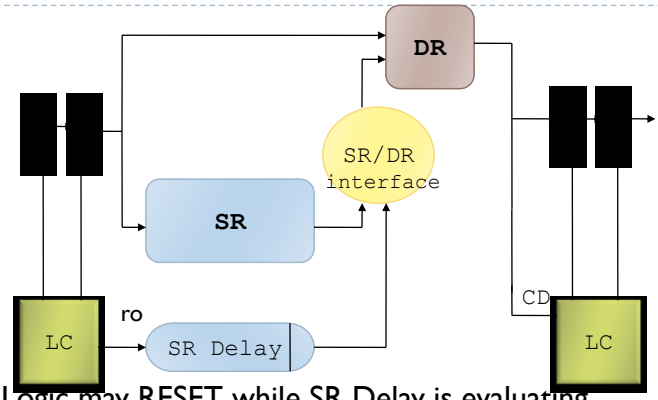


Mixed SRDR Circuits

Mixed SRDR Circuits

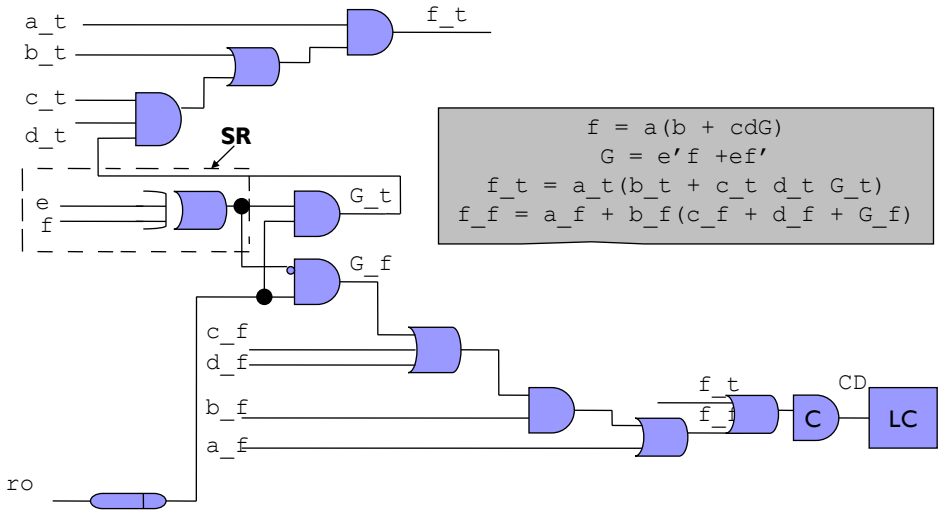
- DR circuits benefits and drawbacks
  - Data-dependent latency (+)
  - Large area overhead (-)
  - Large power overhead (-)
  - NULL (RESET) phase is **slow!!!** (-)
- Why not implemented Mixed SR and DR Logic?
  - Slice C.L. circuit's Logic Levels vertically
  - Use conventional Boolean (SR) logic for first logic levels of C.L.
    - Traversed anyway by most exercised circuit paths
  - Use DR logic only for deep logic levels
    - Exhibit data-dependent elasticity
- ***Hide NULL phase of DR Logic***
  - **Overlap with SR evaluation!!!**

Mixed SRDR Circuit Architecture



- DR Logic may RESET while SR Delay is evaluating
  - Hide DR RESET Overhead!!!

Mixed SRDR Circuit Example



f = a(b + cdG)
G = e'f +ef'
f_t = a_t(b_t + c_t d_t G_t)
f_f = a_f + b_f(c_f + d_f + G_f)