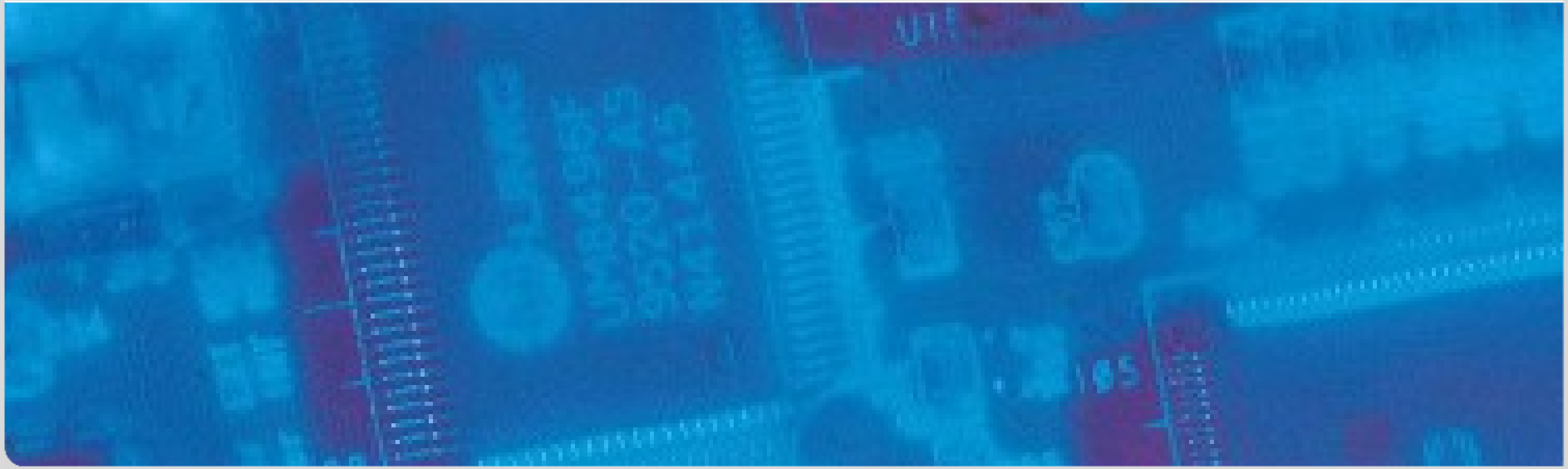
Karlsruhe Institute of Technology

Low Power Design

Volker Wenzel on behalf of Prof. Dr. Jörg Henkel
Summer Term 2016

CES – Chair for Embedded Systems



Lecture Slides

Overview Low Power Design Lecture

- Introduction and Energy/Power Sources (1)
- Energy/Power Sources(2): Solar Energy Harvesting
- Battery Modeling – Part 1
- Battery Modeling – Part 2
- Hardware power optimization and estimation – Part 1
- Hardware power optimization and estimation – Part 2
- Hardware power optimization and estimation – Part 3
- Low Power Software and Compiler
- Thermal Management – Part 1
- Thermal Management – Part 2
- Aging Mechanisms in integrated circuits
- Lab Meeting

Overview for today

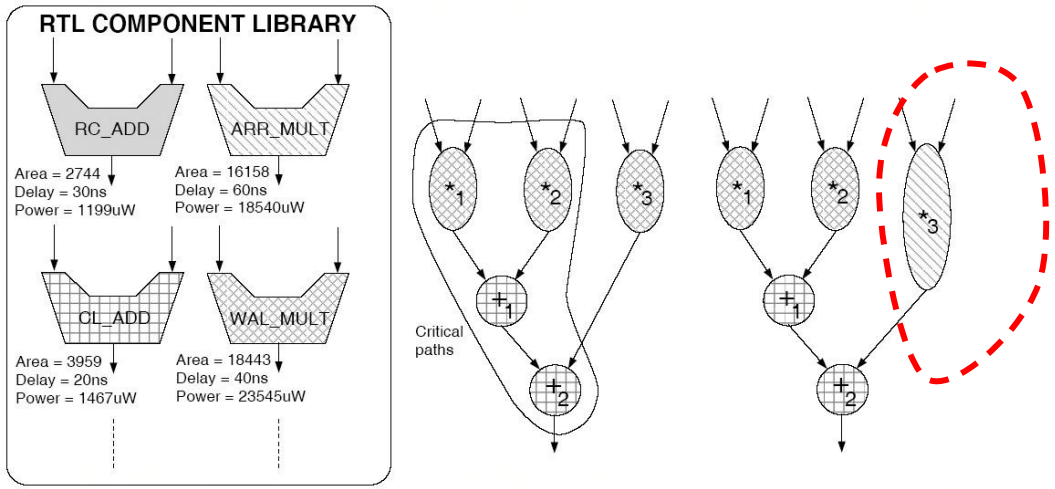
- Recap: Module selection
- Peak Power
- Resource Sharing
- DFG restructuring
- Glitch power reduction
- Clock Gating

Recap: Module Selection for Low Power

Recap: Module Selection for Low Power

- Module Selection
  - process of mapping operations from the CDFG to component templates of RTL library
  - optimize power by chosing most suitable component from template library
- Example: „+“-Operation
  - implementation alternatives:
    - **Ripple-Carry Adder** (slow, but more efficient in switched capacitance)
    - **Carry-Lookahead Adder** (faster, but less efficient in switched capacitance)
    - ...
- Similar tradeoffs exist for other operations

Recap: Module Selection for Low Power (cont'd)

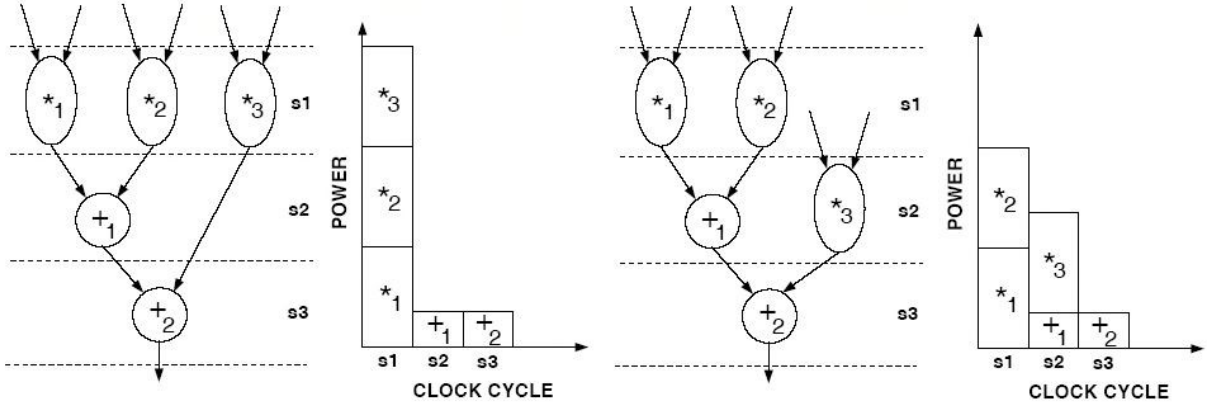


(src:[Anand98])

- Every op in CDFG initially mapped to fastest component due to performance constraint
- Not necessary to map all ops to fastest component; better focus on critical path
- Exploit **slack** in off-critical path ops to select slower functional units that may have a better efficiency in switched capacitance.
- Important to have a large module library with distinct switching capacity efficiencies and performance characteristics
- components might operate at different V_{dd}

Peak Power

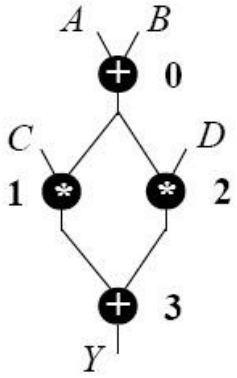
Peak Power

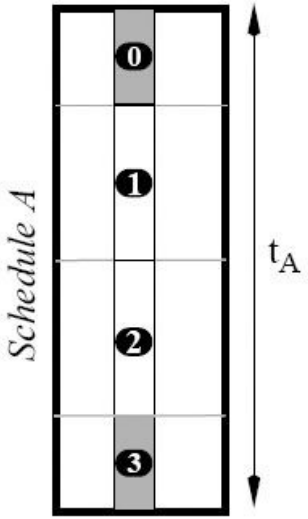


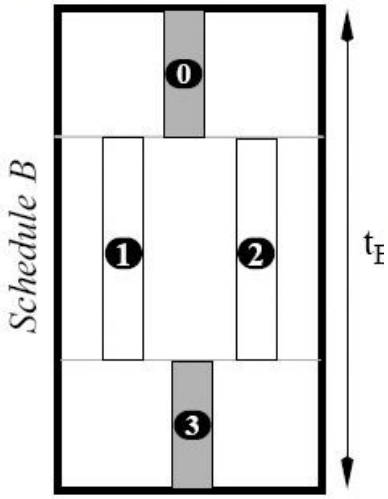
(src: [Anand98])

- Example: two possible schedules of a given CDFG
  - assumption: same power for each operation
  - ASAP schedule results in high peak power
- Slack may be used to reduce peak power without sacrificing any performance

Peak Power (cont'd)







(src:[Knight])

		3.3V	5V
RCA	Total Delay (ns)	30.0	20.0
	Power (mW)	5.4	22.7
CLA	Total Delay (ns)	20.0	10.0
	Power (mW)	10.5	37.3
QBS	Total Delay (ns)	640.0	640.0
	Power (mW)	11.00	28.5
BOOTH	Total Delay (ns)	320.0	160.0
	Power (mW)	12.7	84.0
ARR16	Total Delay (ns)	330.0	170.0
	Power (mW)	24.6	101.3
ARR32	Total Delay (ns)	160.0	100.0
	Power (mW)	143.1	295.6

- Assumption: **1**,**2** and **0**,**3** may pair-wise share a module.
- Resulting delays t_A,t_B

$$t_A = 2 \times t_{ADD} + 2 \times t_{MUL} \qquad t_B = 2 \times t_{ADD} + t_{MUL}$$

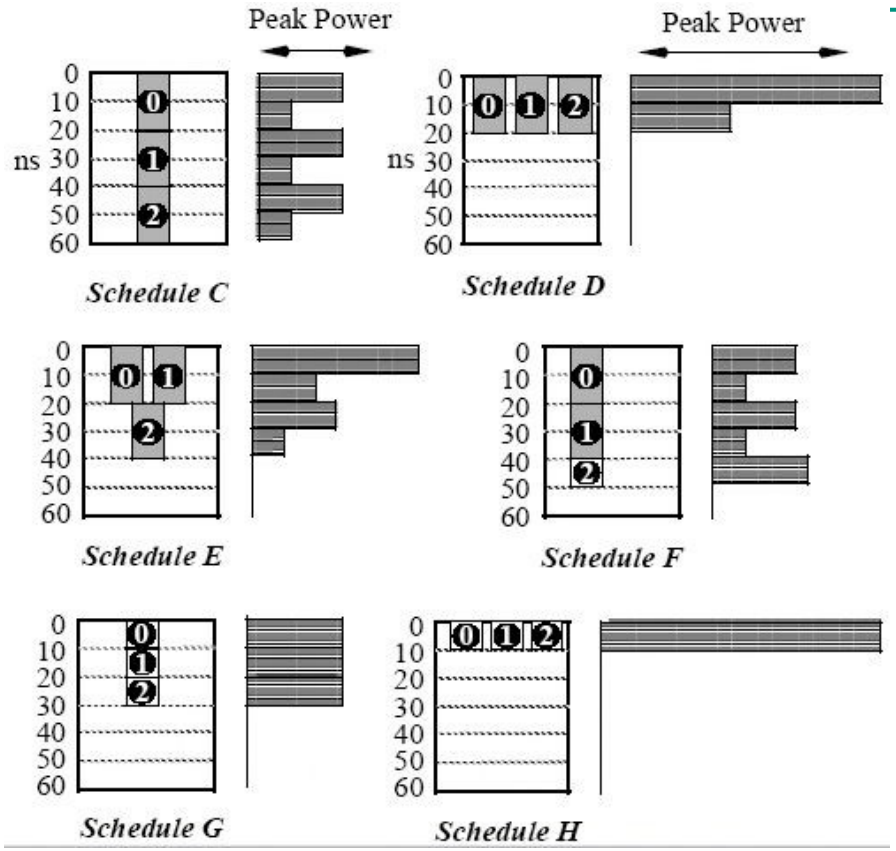
- Average power consumption for the two schedules

$$P_A = \frac{2 \times P_{ADD} \times t_{ADD} + 2 \times P_{MUL} \times t_{MUL}}{t_A} \qquad P_B = \frac{2 \times P_{ADD} \times t_{ADD} + 2 \times P_{MUL} \times t_{MUL}}{t_B}$$

  - Example: t_A=1,300ns; t_B=660ns, P_A=28.6mW, P_B=56.4mW

Note: average power improved considerably, but energy is approx. the same!

Peak Power (cont'd)



Schedule	Delay (ns)	Peak Power (mW)	Average Power at min. delay (mW)	Average power at 60 ns (mW)
C	60	32.7	22.7	22.7
D	20	98.1	68.1	22.7
E	40	65.4	34.1	22.7
F	50	37.3	25.6	21.3
G	30	37.3	37.3	18.7
H	10	111.9	112.0	18.7

Min Delay: refers to when the operations are actually completed, whereas the other case always assumes a 60ns window.

- Window is 60ns
- 'C' uses multicycling and no parallelism, has lowest peak power and is slowest
- 'H' is fastest, has highest peak power but comparable average power (according to the window)
- Note: optimizing for peak power or for average power are completely distinct tasks.

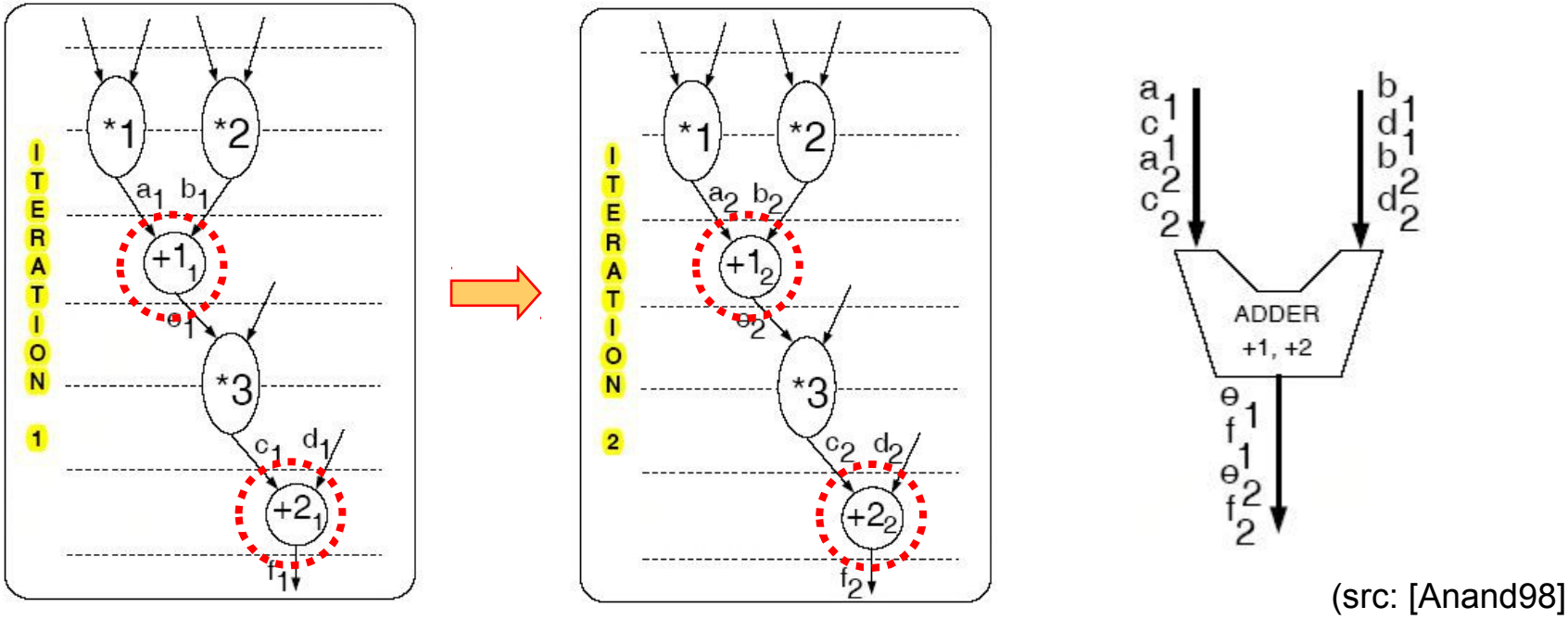
(src: [Knight])

Resource Sharing

Resource Sharing for Low Power

- Resource Sharing is
  - Mapping OPs and variables in CDFG to FUs, registers, etc.
  - Defining interconnection between them to form RTL implementation
  - Mapping from function to structure
  - Directly impacts power consumption by determining switching activity at various signals, buses, wires, macro blocks, etc.
- Observation:
  - Result of resource sharing of variables' values are time multiplexed registers
  - Values that appear as input operands of OPs are time-multiplexed to appear at inputs of FUs
  - Values that are transferred between FUs are sequenced to appear on interconnect units (buses and multiplexers)
  - Word-level temporal correlations of values on data path signals are determined by the correlations among variables and input operands of operations that are grouped together during resource sharing
  - Word-level correlation determine bit-level switching activity

Resource Sharing for Low Power (cont'd) – Exploiting Signal Correlation –



(src: [Anand98])

- Focus on **+1** and **+2** operations
- Two consecutive iterations of the DFG are shown: **+1**$_1$, **+2**$_1$, **+1**$_2$, **+2**$_2$
- Values seen at adder inputs are: (a1, b1), (c1,d1), (a2, b2), (c2, d2)
- What is the switching activity at the adder inputs determined by?

Resource Sharing for Low Power (cont'd) – Exploiting Signal Correlation –

- Switching activity is determined by
  - Intra-iteration effects

    Hamming Distance between the values of a1 and c1 (also b1 and d1) in the first iteration and a2 and c2 (also b2 and d2) in the second iteration
  - Inter-iteration effects

    Hamming Distance between the values of c1 and a2 (also d1 and b2)

- Idea: Exploit correlations between variables in behavioral description to minimize switched capacitance at RTL level

Resource Sharing for Low Power (cont'd) – Exploiting Signal Correlation –

Scenario 1: Application characterized by slow varying inputs

- inputs are highly temporally self-correlated (e.g. DSP applications)
- internal variables also correlated
- temporal correlations are typically inter-iteration correlations
- architecture with little or no hardware sharing might be better
- hardware sharing could destroy temporal correlations → unnecessary switching activity

Scenario 2: Value assumed by an input in an iteration correlated with value assumed by other inputs in the same iteration

- e.g. correlated sound tracks in high-quality audio applications fed to different speakers

Resource Sharing for Low Power (cont'd) – Exploiting Signal Correlation –

Scenario 3: Functional relationship between signals imposed by the correlation leads to correlation

- e.g. variable that represents result of operation might be correlated with the variables that represent the input operands of the operation

op	$+$	$*$	AND	OR
correlation	0.500	0.617	0.75	0.75

(src: [Anand98])

Resource Sharing for Low Power (cont'd) – Exploiting Signal Regularity –

- Regularity: repeated occurrence of computational patterns within an algorithm
- Idea: Exploit regularity to reduce interconnect power
  - Detect instances of repetitive patterns in the computation and resource sharing by
  - reusing same interconnect structure for as many instances of computation as possible

Resource Sharing for Low Power (cont'd) – Exploiting Signal Regularity –

Left: data flow graph + implementation

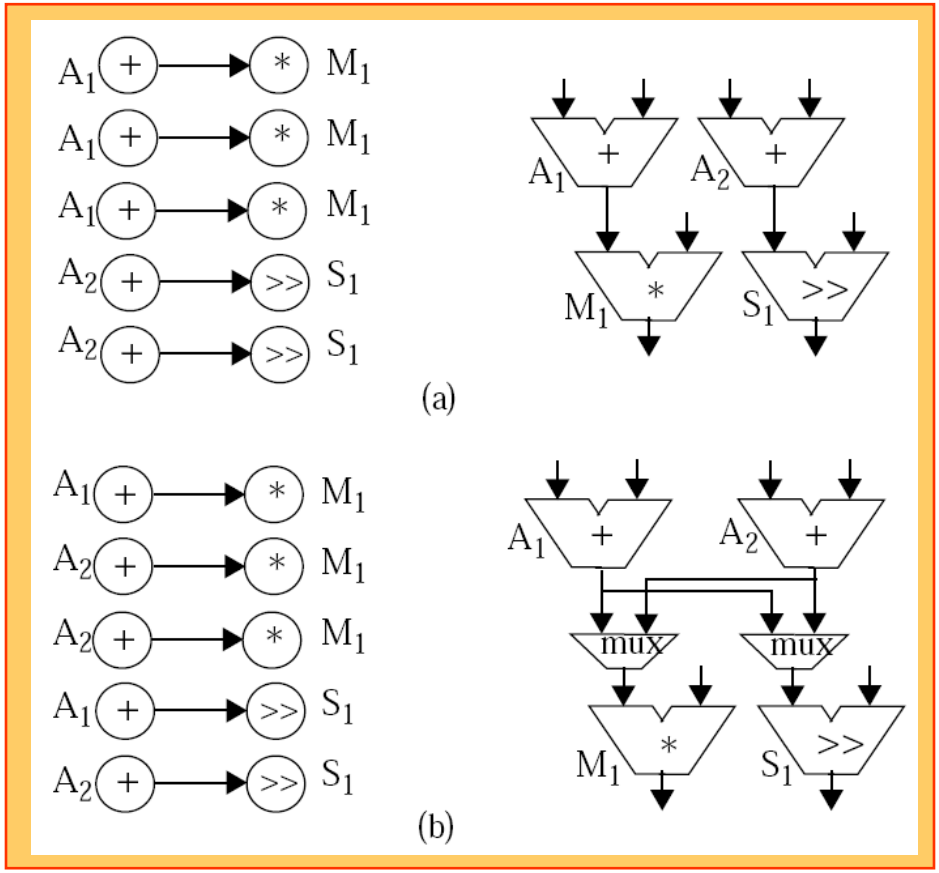
b) Problem: for A2 → M1, A2 → M1, A1 → S1 multiplexers are needed

a) same data flow graph, but mapping to data path does not require multiplexers

Power consumption overhead:

- More fan outs → larger interconects → more switched capacitance
- Overhead from multiplexers (and probably drivers) leads to more switching activity

Conclusion: b) does not preserve regularity, a) does



(src: [Mehra])

Resource sharing for LP (cont'd) - exploiting signal regularity -

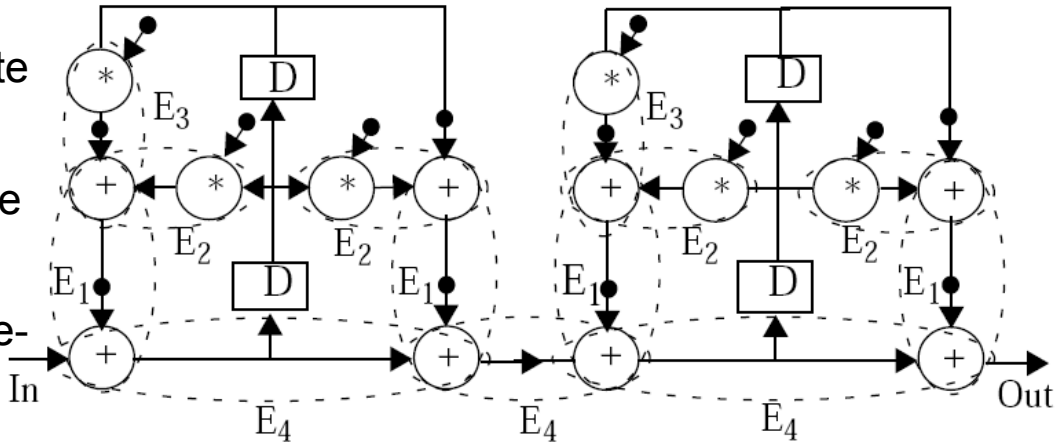
- Idea: defining E-instances: a pair of nodes connected by an edge

  E-template: type of an E-instances classified by type of input/output port Ex: (add->add.right) means a template with an adder where output maps to right input of another adder

  E-coverage: # of instances of that type divided by total # of edges in graph

  task here: using E-templates in synthesis as to minimize power with e-coverage as quality measure through regular assignment

- **Ex 2**: a fourth-order cascade filter
- Disadvantages?
  - may require more hardware units since sufficient E-templates need to be provided -> under circumstances power savings come at cost of more hardware



(src: [Mehra])

E-template	Coverage
E1 (add → add.right)	4/26
E2 (mult → add.left)	4/26
E3 (mult → add.right)	2/26
E4 (add → add.left)	3/26

Resource sharing for LP (cont'd)
- exploiting signal regularity -

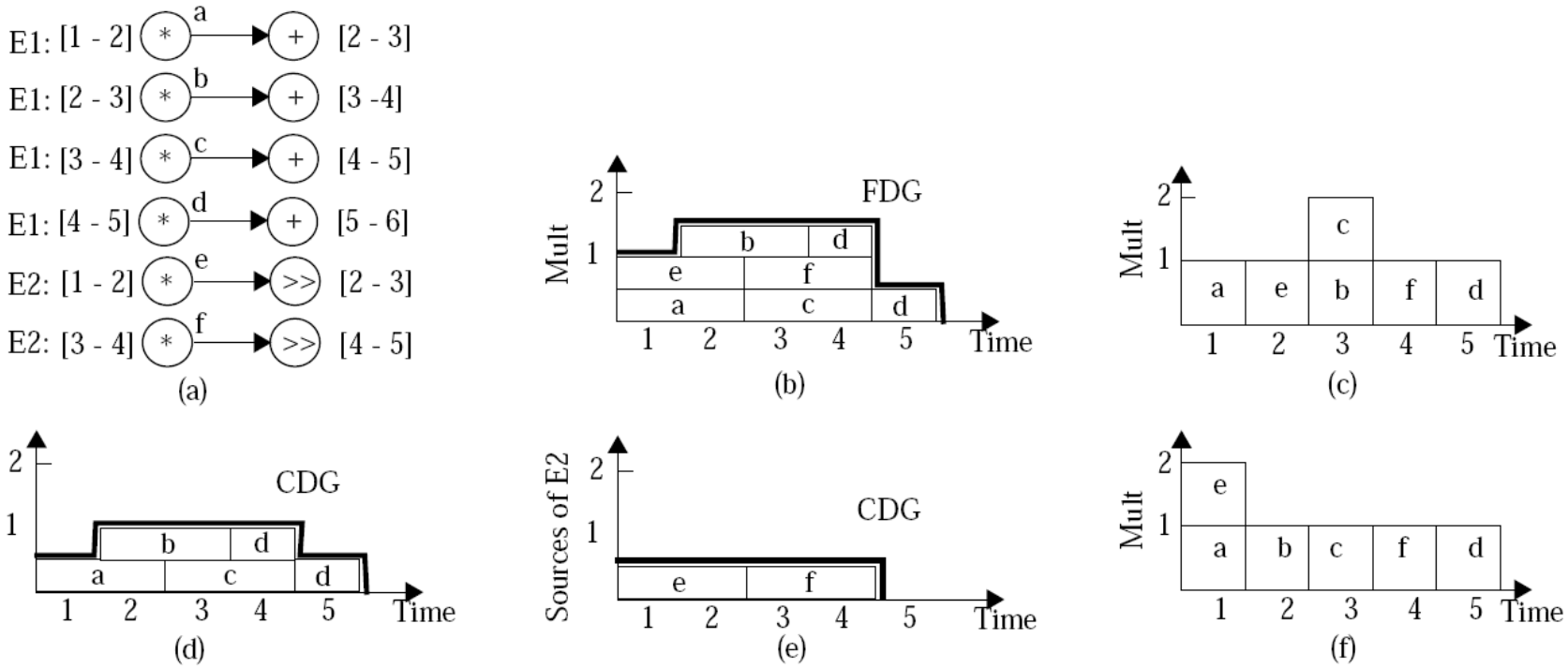


Figure 3. The effect of using connection distribution graphs, (a) Instances of two E-templates with their ASAP and ALAP times, (b) Initial FDG for multiply operations, (c) Final distribution graph using only FDGs, (d, e) Initial source CDGs of the two E-templates, (f) Final distribution graphs using FDGs and CDGs.

(src: [Mehra])

- Example: E-template-based scheduling

DFG restructuring for Low Power

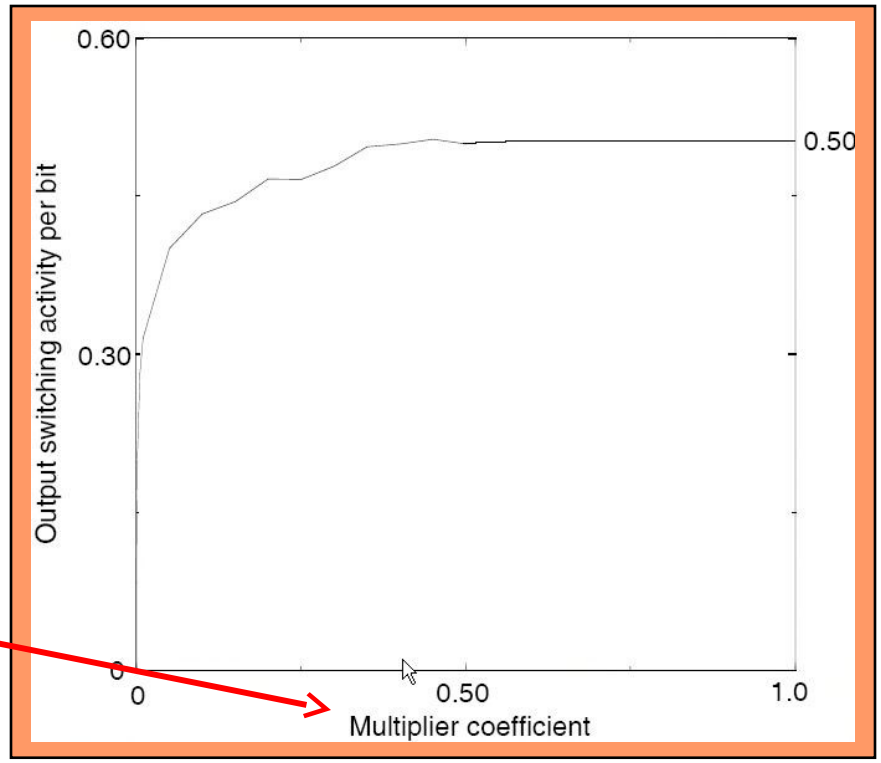
DFG restructuring for LP

- A typical DSP operation: constant multiplication and addition: Y = A * X (e.g. as part of an IIR filter)
- Assumptions:
  - m-bit data value X multiplied by m-bit constant A
- Experiment:
  - Applying random values of X for varying values of A
  - X and A are represented as two's complement
  - A 0.0 … 1.0 normalized
  - Observing average switching activity per bit at multiplier output
  - Result: next slide

DFG restructuring for LP (cont'd)

- **Observation:**
  - When constant value A is '0' → no switching activity (as expected)
  - When A is 1.0, output switching activity is equal to switching activity of X
  - In between: it is monotonically increasing

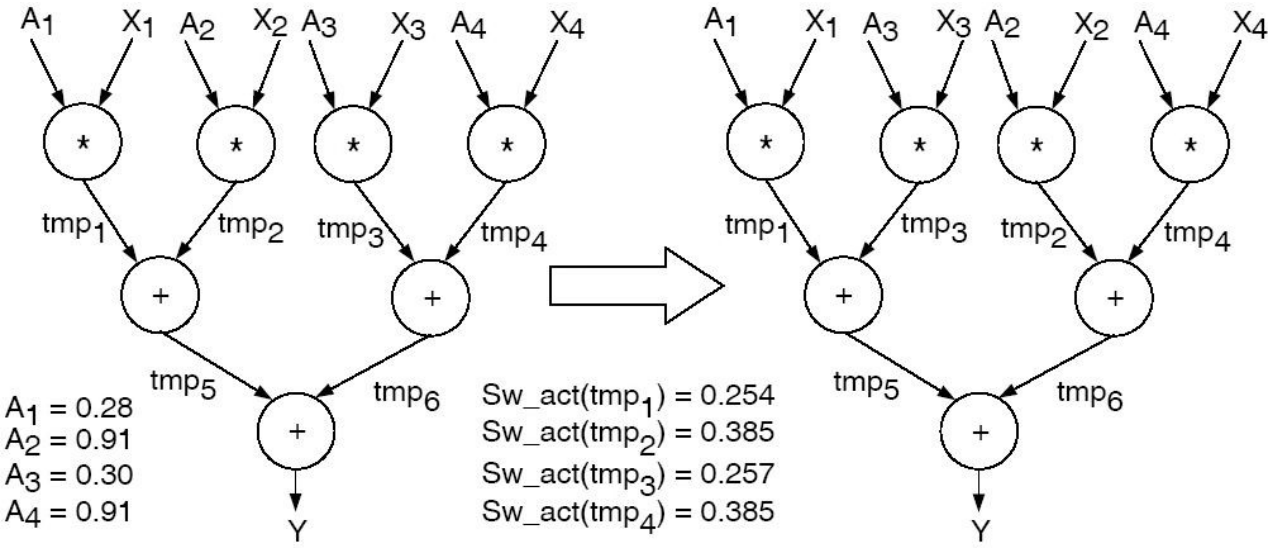
- Example
  - An adder: two m-bit data values X1, X2. It can be shown that:



(src: [Anand98])

$$Sw\_act(Y) = \max(Sw\_act(X_1), SW\_act(X_2))$$

DFG restructuring for LP (cont'd)



- **Example:** linear time-invariant signal processing system (e.g. IIR filter)

$$Y = \sum_{i=1}^{n} A_i . X_i$$

(src: [Anand98])

  - Left figure

$$\begin{aligned} Sw\_act(tmp_5) &= \max(Sw\_act(tmp_1), Sw\_act(tmp_2)) = 0.385 \\ Sw\_act(tmp_6) &= \max(Sw\_act(tmp_3), Sw\_act(tmp_4)) = 0.385 \\ Sw\_act(Y) &= \max(Sw\_act(tmp_5), Sw\_act(tmp_6)) = 0.385 \end{aligned}$$

  - Right figur

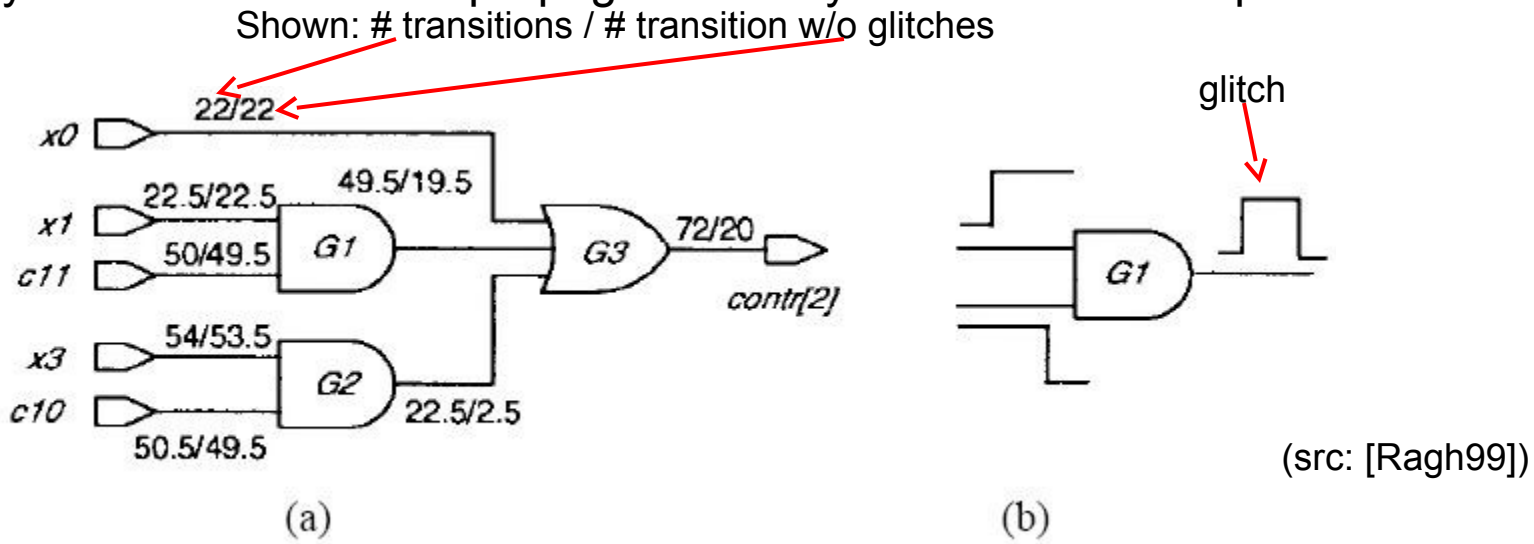
$$\begin{aligned} Sw\_act(tmp_5) &= \max(Sw\_act(tmp_1), Sw\_act(tmp_3)) = 0.257 \\ Sw\_act(tmp_6) &= \max(Sw\_act(tmp_2), Sw\_act(tmp_4)) = 0.385 \\ Sw\_act(Y) &= \max(Sw\_act(tmp_5), Sw\_act(tmp_6)) = 0.385 \end{aligned}$$

Glitch Power Reduction

Glitch power reduction

What is “glitch power”?

- Power consumption that is related to hazards i.e. temporary values at the input/output of gates that cannot be explained when considering a truth table only. It is due to different propagation delays in combinatorial paths



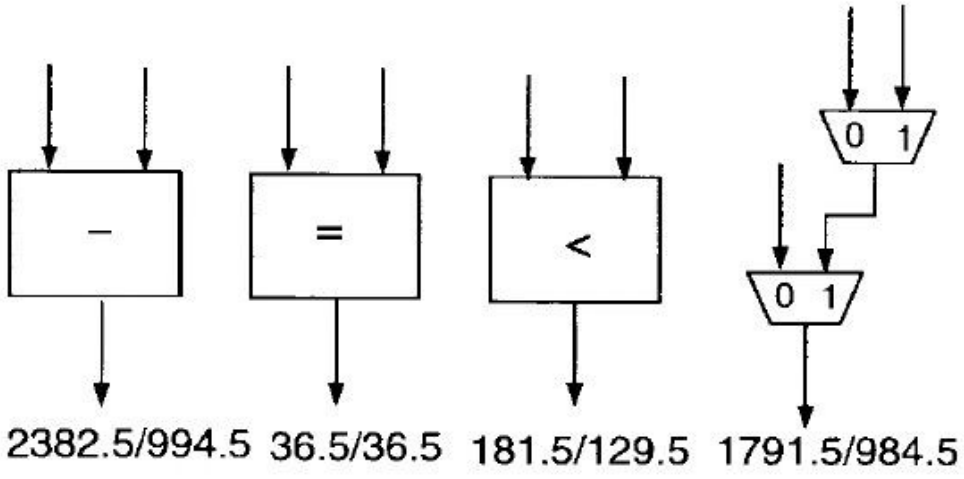
(src: [Ragh99])

- Analysis of the above example unveiled:
  - A rising transition on signal x1 was frequently accompanied by a falling transition on c11. Thus, the rising transition on x1 and the falling transition on c11 are highly correlated.
  - Transitions on signal x1 arrive earlier than transitions on signal c11 due to: a) non-balanced paths, b) wiring delays.

Glitch power reduction (cont'd)

- **In general**, **glitches are generated** at the **control signals** due to the simultaneous presence of the following two conditions. *1)* **Functional**: Correlation between rising and falling transitions at two or more signals that feed a gate. *2)* **Temporal**: The controlling to non-controlling transition arrives earlier at the gate's input (see example last slide)
- Note: glitchy signal propagates (and therefore consumes even more power at other gates in the circuitry => try to eliminate the glitch as close to the location of first occurrence (i.e. where it is generated) as possible
- Glitches may also be generated by **data path blocks**: examples

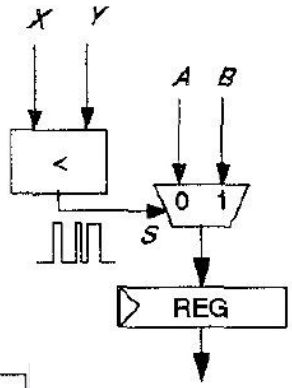
Assumption: input signals are considered to be glitch -free and to arrive simultaneously → glitches reported are generated by the respective block



Note: comparator '=' is glitch-free -> seems to indicate that the logic inside is well-balanced

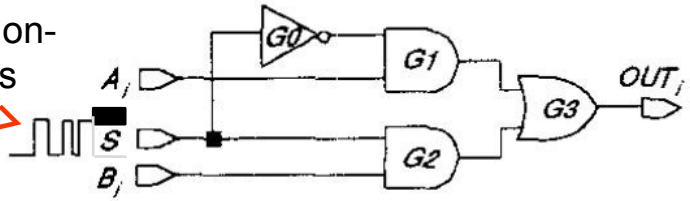
Glitch power reduction (cont'd)

- Example: multiplexer of 2 8-bit words is controlled by a comparator "<"
  - Assumption: comparator generates glitches; A,B are glitch-free



- Shown: a bit slice of words A and B

Glitchy and non-glitchy signals



A	B	W/O Gl.	With Gl.
0	0	0.5	0.5
0	1	1.0	3.0
1	0	1.0	3.0
1	1	0.5	3.5

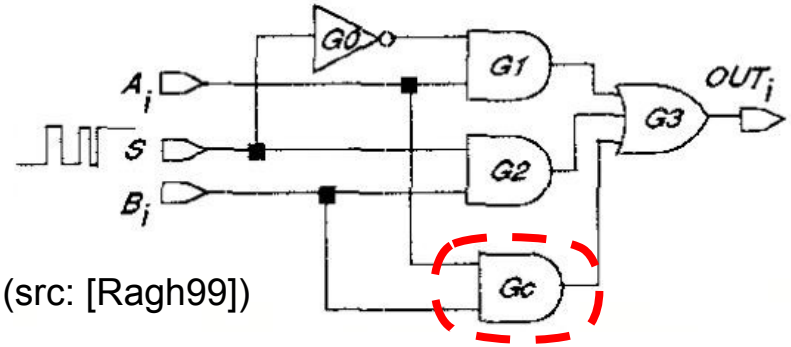
In table: # of transitions w/o glitches, total # of transitions

(src: [Ragh99])

- Discussion (notation: $<A_i,B_i>$)
  - <0,0> : glitch cannot propagate
  - <0,1>, <1,0> : glitch propagates at G1 for <1,0> and at G2 for <0,1>
  - <1,1> : glitch always propagates
- How to prevent glitches ?
  - For example: use **spatial correlations**

Glitch power reduction (cont'd)

- Spatial correlation: observation: value of S is irrelevant in case <1,1> (anyway '1' at OUT_i) => insert gate G_c => propagation of glitch on S is prohibited
- Result:



(src: [Ragh99])

A	B	[redacted]	[redacted]
0	0	0.5	0.5
0	1	1.0	3.0
1	0	1.0	3.0
1	1	0.5	0.5

- Q: why can't just a buffer be inserted between S and input of G2 (idea here: the glitches compensate and therefore eliminate each other) ?
- These and other techniques need to be built into synthesis tools and component libraries in order to prevent glitches in the first place
- More techniques to prevent glitches during synthesis can be found in [Ragh99]
- Problem with glitch reduction techniques:
  - Power overhead of additional gates (might reduce gains obtained through reduced number of glitches)
  - Increased power of existing gates due to more inputs (example above: G_3 has 3 instead of 2 inputs)
  - ...

Clock Gating

Clock gating

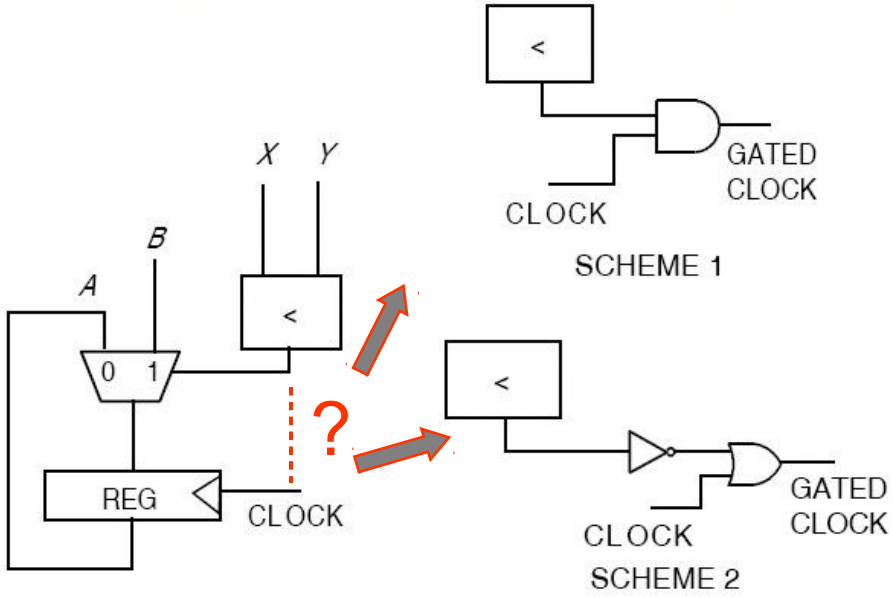
- What is clock gating?

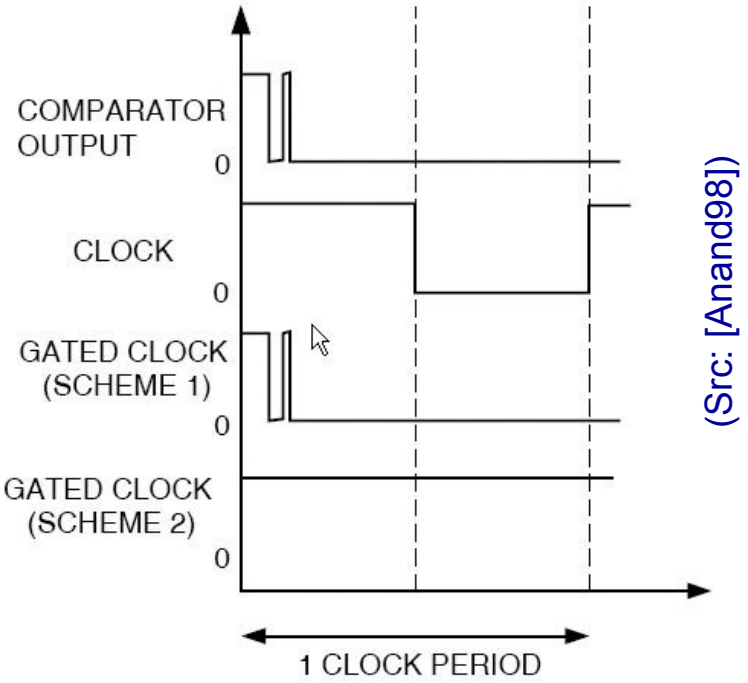
Idea: suppress or disable transitions from propagating to parts of the clock network under specific conditions that are determined by the clock gating circuitry

- How can clock gating result in power savings?
  - Reduced capacitive switching in the clock network like
    - clock buffers
    - interconnect of the clock network
    - latches/registers that are fed by the clock signal
  - Also:
    - may prevent storage elements from loading unnecessary new values and thus saving power

(Src: [Anand98])

Clock gating (cont'd)





- Register re-loads previous value when comparator output is '0' => transition at the clock input to register can be suppressed and transitions can be spared
  - Scheme 1: register clock input would be forced to '0' when comp is '0' (desired)
  - Scheme 2: register clock input is forced to '1' when comparator output evaluates to '0' (desired)

Scheme 1: does not work since comp output is not stable before clock edge rises

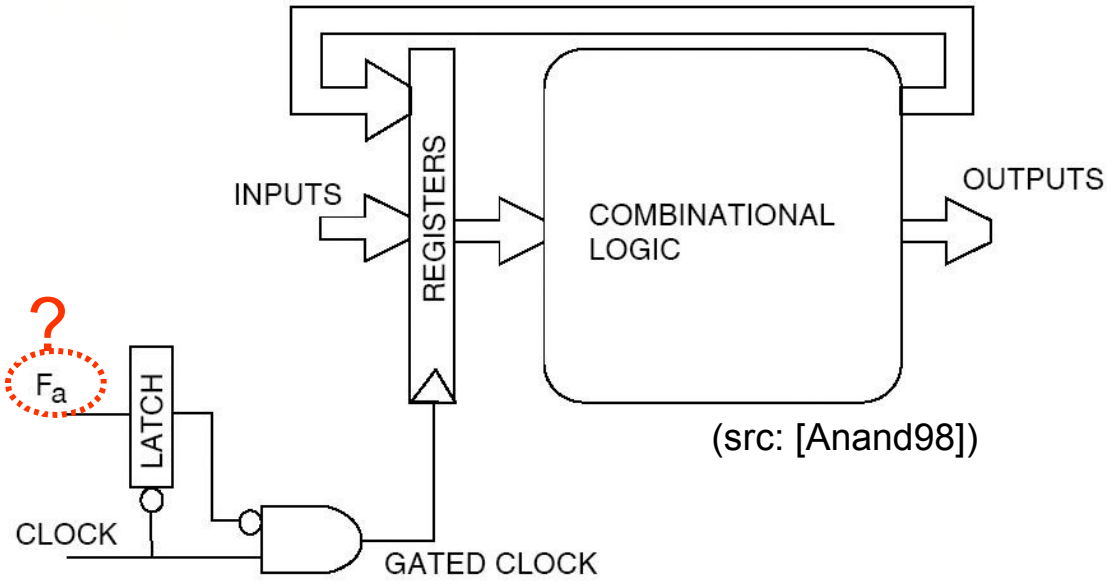
Scheme 2: OK (as long as gating condition stabilizes before clock does '0' -> '1')

Clock gating (cont'd)

- Typically: a) existing signals in the circuitry may be used for gating parts of the clock network or, b) signals from previous clock may be used (in that case those signal values need to be stored in latches)
- Example:
  - Decode stage of a micro-processor pipeline can be used to clock-gate later stages
- In other case:
  - Additional circuitry needs to be added
- Pitfalls and overheads:
  - Introducing additional gates in clock tree may lead to an increase in clock delay and clock skew
  - Ensure that gating clocks does not introduce glitches, otherwise: malfunction due to spurious loading of registers
  - Circuits with gated clock introduce additional complexity to synthesis and analysis tools

Clock gating (cont'd)

- Consideration: identify conditions when next state and primary output conditions do not change
- Gating the clock on its roots (e.g. for whole circuitry)
  - → eliminates clock skew problem
- Added circuitry may incur additional power etc.
  - → try to detect subset of idle condition at low overhead



(src: [Anand98])

- Idea: automated gated-clock synthesis (architecture, see above)
  - Synthesizing an activation function F_a:
    - goes to '1' when clock needs to be stopped
    - Latch L ensures that glitches are not propagated to clock signal
    - AND gate suppresses eventually clock for whole circuitry

Clock gating (cont'd) - synthesizing F_a -

- Given: a Moore FSM $(PI, PO, S, s0, \delta, \lambda)$
  - PI set of inputs
  - PO set of outputs
  - S set of states
  - $s0$ initial states
  - δ next-state function
  - λ Output function

Note: for Moore FSM output is a function only of the current state and not of input variables.
A self-loop in state transition graph (STG) corresponds to an idle condition => condition where clock to FSM register can be suppressed

State s_i with self-loop function

$Self_{si} : PI \rightarrow \{0, 1\}$ such that $Self_{si}(pi) = 1$

iff $\delta(x, si) = si,\ pi \in PI$

x_i – decoded state variable corresponding to s_i; x_i = 1 iff FSM is in state s_i

$Self_{si}$ captures the set of input conditions under which the self-loop of state s_i is traversed

(Src: [Anand98])

Activation function:

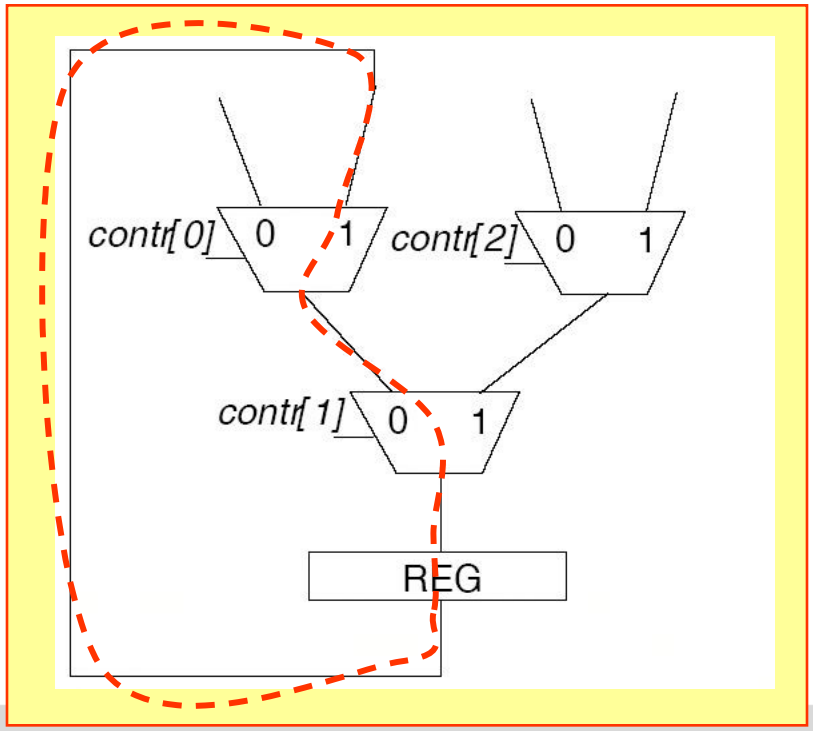
$$F_a = \sum_{i=0,\ldots,|S|-1} Self_{si}.xi$$

Note: activation function might be complex

Clock gating for data paths

- Often in data paths: output of register is fed back as one of the data inputs through, for example, a multiplexer network
- Task: find condition under which this is the case by traversing the path through the multiplexer network

(Src: [Anand98])



The gating condition for the clock input is:

$$contr[0] \cdot \overline{contr[1]}$$

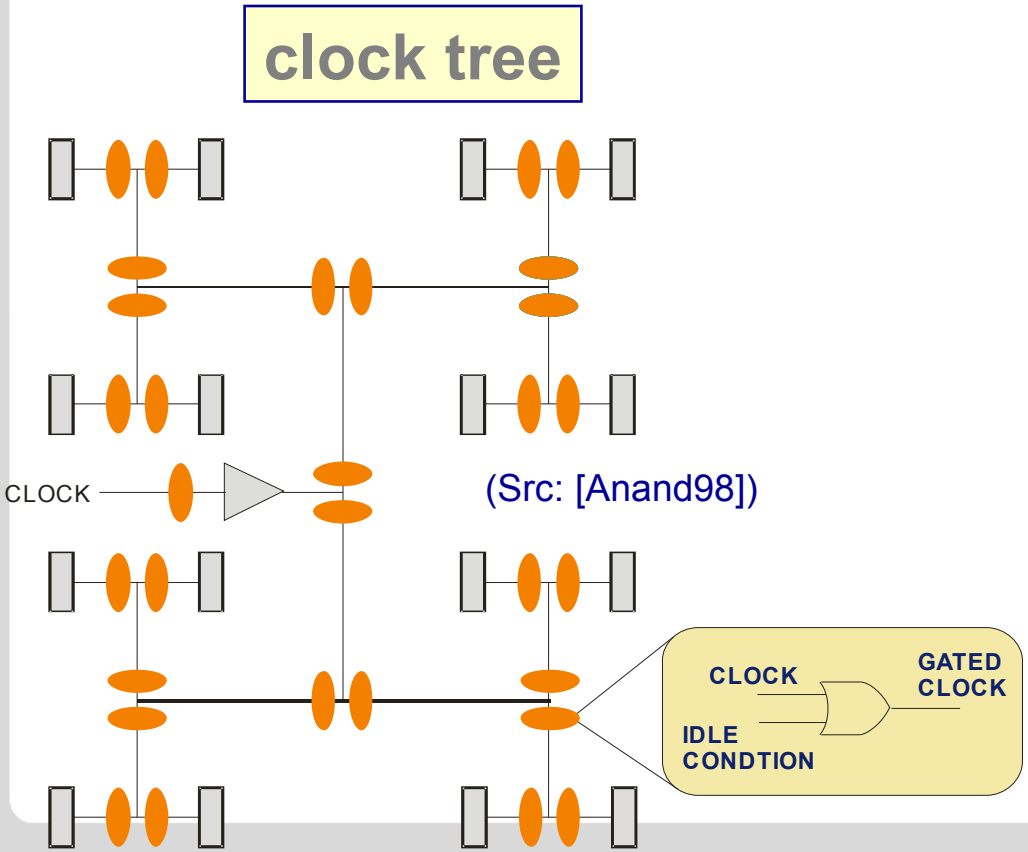
Note: select signals are already present in network => only invert (if necessary) and conjunction need to be provided for gating condition

Caution: strategy does not guarantee timing requirements (i.e. gating condition should stabilize before clk goes '1'->'0')

In order to avoid slower clocking: derive reduced gating condition

Clock gating for data paths (cont'd)

- Shown: a clock tree that has gated clock conditions at various points (levels)



Tradeoff:

- Disabling clock at higher level in the tree =>

a larger capacitance (sum of all smaller ones) is prevented from switching

- But: clock transition at certain level can only be suppressed if all registers of the certain sub-tree can retain their old values

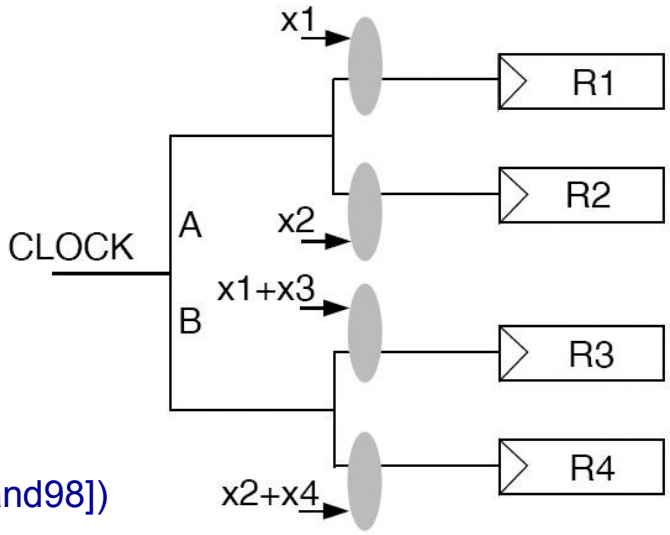
⇒gating condition is satisfied fewer times =>

reduction of # of transitions saved

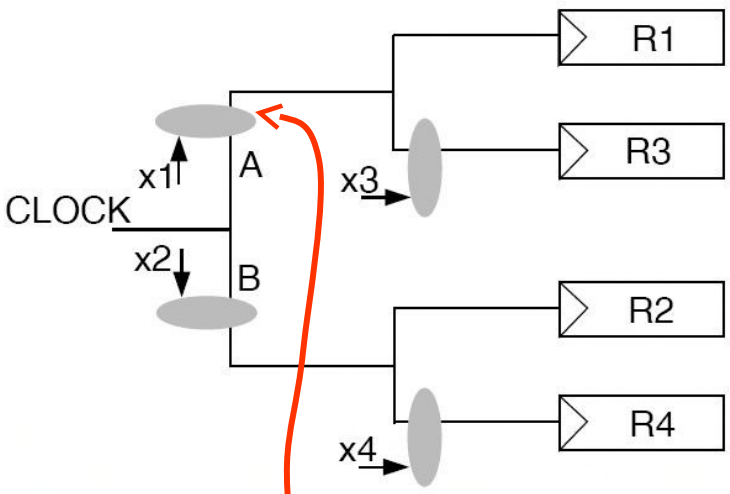
On the other side: when doing at lower level, more transitions could have been saved but that costs more logic (that itself consumes power …)

Clock gating
- clock tree construction -

- Observation: the way the clock tree is constructed has an impact on gating the clock tree (see example):



(Src: [Anand98])



Left: shown a clk tree with four registers R1,R2,R3,R4 and the conditions under which clk tree can be disabled. x1,…,x4 are decoded controller stated variables which are mutually exclusive (none of them can assume a '1'simultaneously)

Observation: R1, R2 are grouped under a clock tree even though their conjunction can never be true => not possible to gate the clock at point "A", for example (similar R3, R4)

Right: gating condition for sub-tree under "A":

$$GC_A = x1 \cdot (x1 + x3) \cdot x1$$

("B" may be grouped similarly)

Advantage: more suited to gated clock since groups of registers with similar or overlapping idle conditions closer together

→ trees can be shut down more efficiently

Clock gating
- multiple clocks -

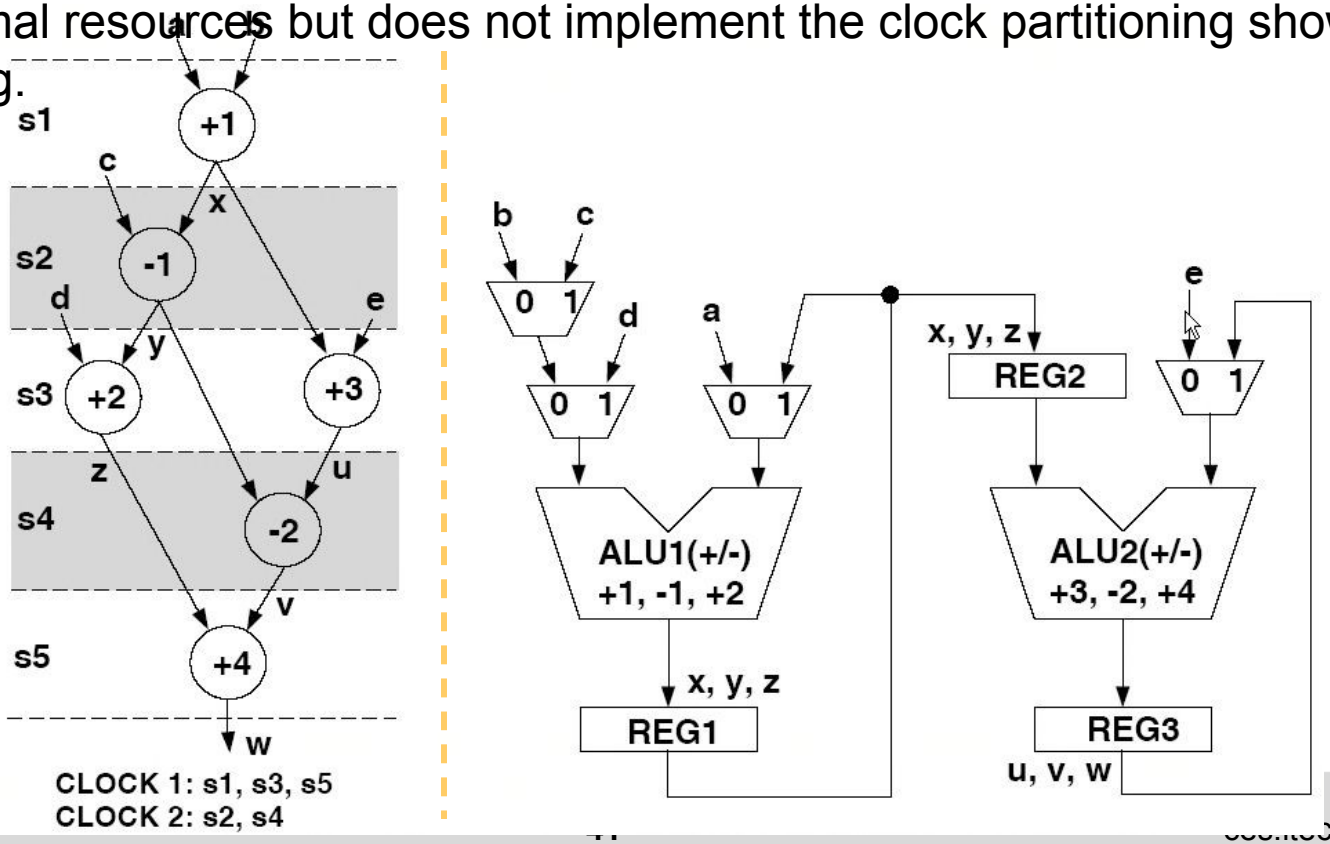
- Observation: some components of a circuit may follow some simple regular patterns. In particular, a component may be idle and active in alternating clock cycles or so
  - => clock gating circuitry needs not necessarily to be data dependent
- Example:
  - A circuit with all registers fed by a single clock; whole capacitance is C and frequency is f
  - Assume: design is partitioned into two parts each fed by clock signals with f/2 and capacitances C1, C2. Power savings can be achieved if:

$$C_1 \frac{f}{2} + C_2 \frac{f}{2} < C f$$

  - $C_1 + C_2 < 2C$ so, circuit needs to be partitioned carefully what should often be possible to achieve
  - Note: - savings here are for clock tree only
    - the f/2 does not result in performance penalties in this example

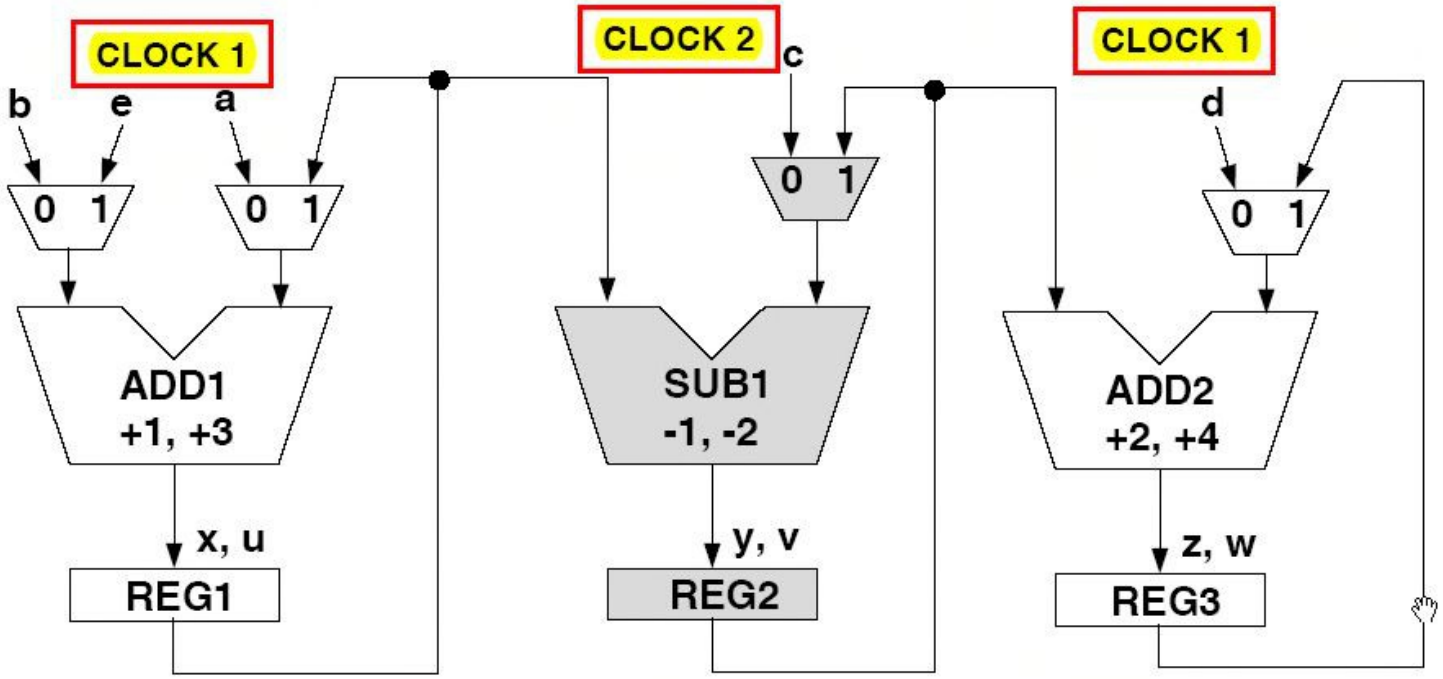
Clock gating
- multiple clocks (cont'd) -

- Idea: using multiple non-overlapping clocks. Example:
  - Scheduled DFG (left fig.): clock cycles of the schedule s1,…,s5 have been assigned to two non-overlapping clock domains, CLOCK1, CLOCK2 in alternating way
  - Right fig: shows single-clock RTL circuit that implements given DFG using minimal resources but does not implement the clock partitioning shown in left fig.



(Src: [Anand98])

Clock gating
- multiple clocks (cont'd) -



(Src: [Anand98])

Shown: RTL circuit that has been implemented with two clocks

Restrictions:

a) an op scheduled in CLOCK1 cannot share an FU with an op in CLOCK2

b) a variable generated in CLOCK1 cannot share an FU with a variable generated in CLOCK2

Why? => 1. ensure that each register can be clocked by either CLOCK1 or CLOCK2

2. data path can be partitioned into two domains such that there is only switching activity in their respective active clock cycles

Clock gating
- some disadvantages -

- Inserting additional gates into the clock tree can lead to an increase in the clock delay and clock skew
- Circuit malfunction due to spurious loading of registers when not taking into consideration that gating logic might introduce glitches
- Increase of complexity to synthesis and analysis tools

Sources

[Heer04] Ch. Herr, U. Schlichtmann, “Ultra-Low-Power Design: Device and logic design approaches”, pp. 1-20, in “Ultra Low-Power Electronics and Design” by Kluwer, 2004.

[Anand98] A. Raghunathan, N.K. Jha, S. Dey, “High-level power analysis and optimization”, Kluwer Academic Publishers,1998.

[Sarraf95] S. Raje, M. Sarrafzadeh, “Variable voltage scheduling”, IEEE/ACM ISLPED 1995. pp. 9-14, 1995.

[Knight] R.S. Martin, J.P. Knight, “Power-Profiler: Optimizing ASIC’s Power Consumption at the behavioral level”, Proc. Of IEEE/ACM Design Automation Conf. (DAC’95), pp.42-47,1995.

[Macii04] E. Macii (Ed.), “Ultra Low-Power Electronics and Design”, Kluwer Academic Publishers, 2004.

[Devadas] Alidina, M.; Monteiro, J.; Devadas, S.; Ghosh, A.; Papefthymiou, M.; “Precomputation-based Sequential Logic Optimization For Low Power”, Computer-Aided Design (ICCAD), 1994., IEEE/ACM International Conference on November 6-10, 1994 Page(s):74 – 81.

[Ragh99] Raghunathan, A.; Dey, S.; Jha, N.K.; “Register transfer level power optimization with emphasis on glitch analysis and reduction”, Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on Volume 18, Issue 8, Page(s):1114 – 1131, Aug. 1999.

[Tivari] Tiwari, V.; Malik, S.; Ashar, P.; “Guarded evaluation: pushing power management to logic synthesis/design”, Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on Volume 17, Issue 10, Page(s):1051 – 1060, Oct. 1998.

[Mehra] R. Mehra, J. Rabaey, “Exploiting Regularity for Low Power Design”, IEEE/ACM Intl’ Conference on Computer Aided Design (ICCAD96), pp. 166-172, 1996.

[Keating] Keating, Michael, David Flynn, Rob Aitken, Alan Gibbons, and Kaijian Shi. Low power methodology manual: for system-on-chip design. Springer Publishing Company, Incorporated, 2007.

[Kim] Kim, N.S.; Austin, T.; Baauw, D.; Mudge, T.; Flautner, K.; Hu, J.S.; Irwin, M.J.; Kandemir, M.; Narayanan, V., "Leakage current: Moore's law meets static power," Computer , vol.36, no.12, pp.68,75, Dec. 2003