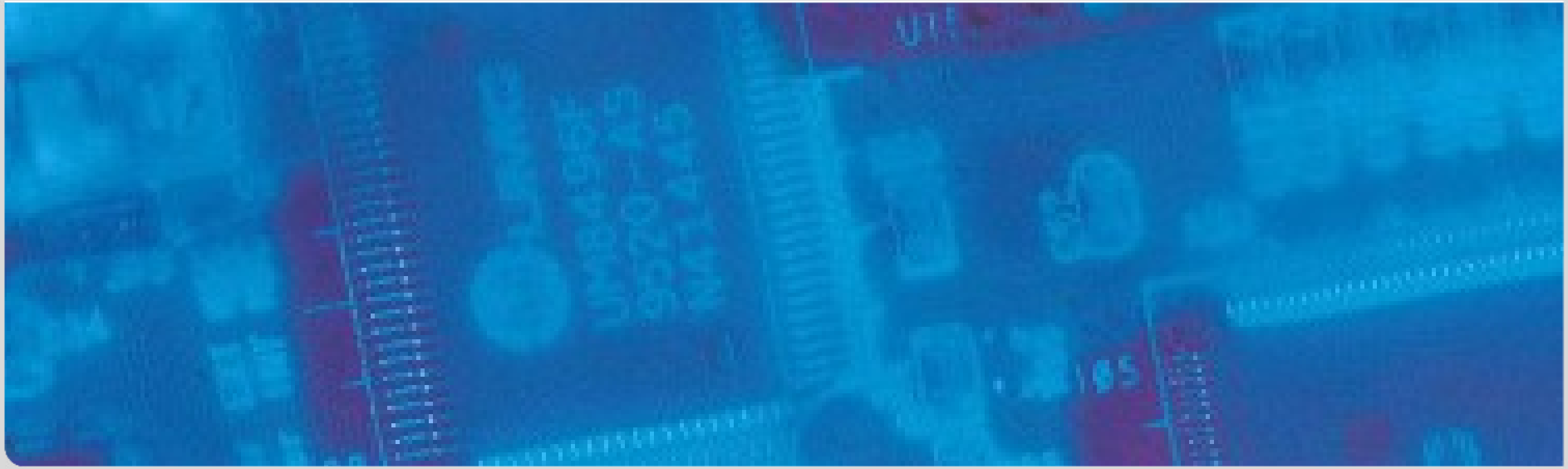
Karlsruhe Institute of Technology

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: Clock Gating
- Power Gating
- Pre-Computation
- Reducing Power by Scheduling
- Operand Isolation
- Register Sharing
- Reducing Power through the Controller

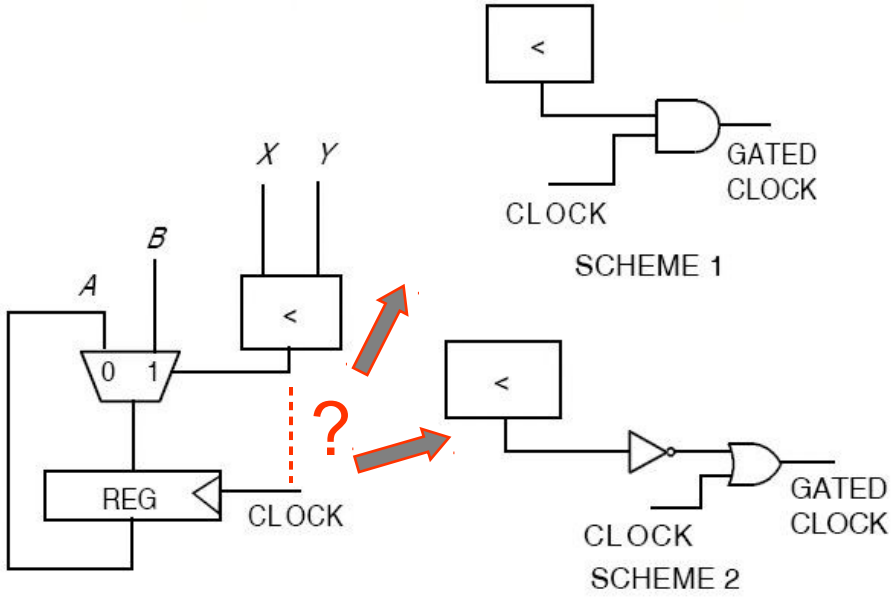
Recap: Clock gating

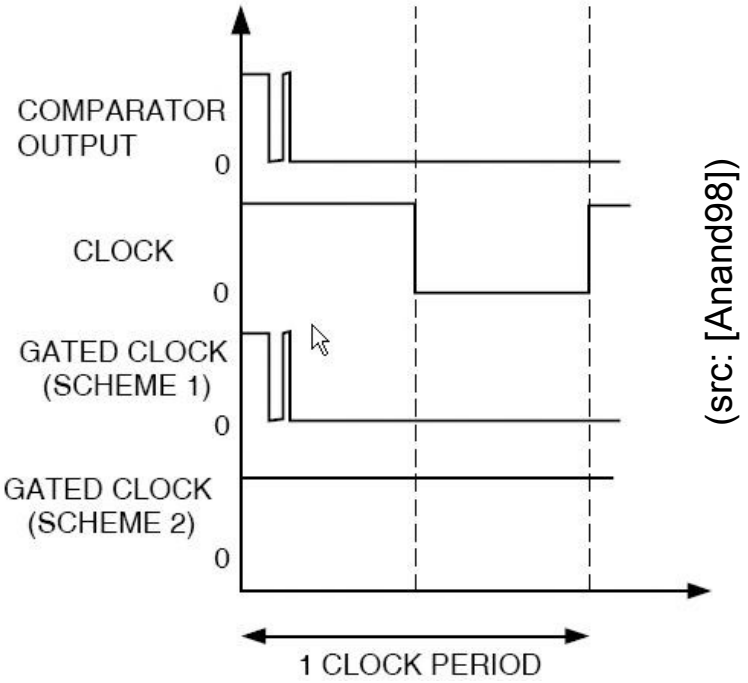
- What is clock gating?

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

Recap: 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

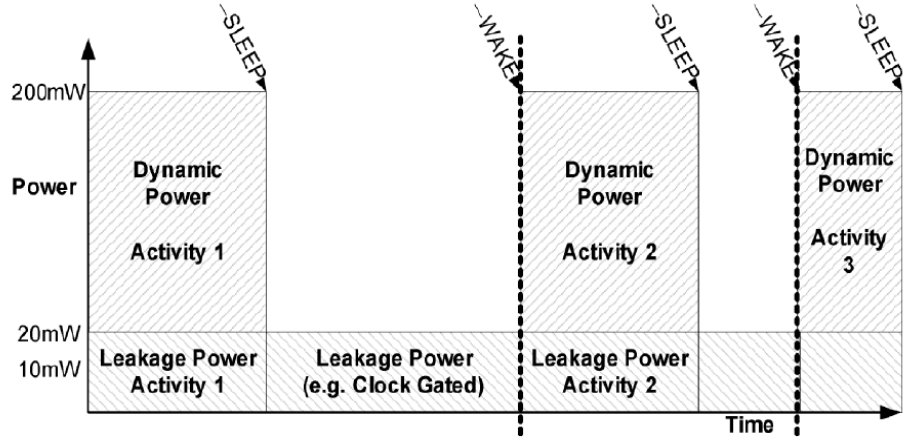
Scheme 2: OK (as long as gating condition stabilizes before clock does '0' -> '1')

Power Gating

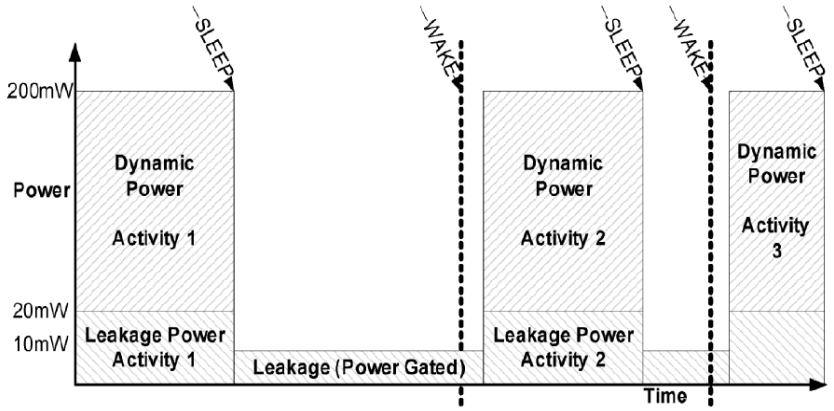
Power Gating

- Power Gating:
  - Bring both V_{dd} and ground to the same voltage potential when circuit idle
- Power savings:
  - Reduced potential greatly minimizes leakage currents
- Disadvantages:
  - slower than clock gating
  - circuit state lost (remedy: retention registers)

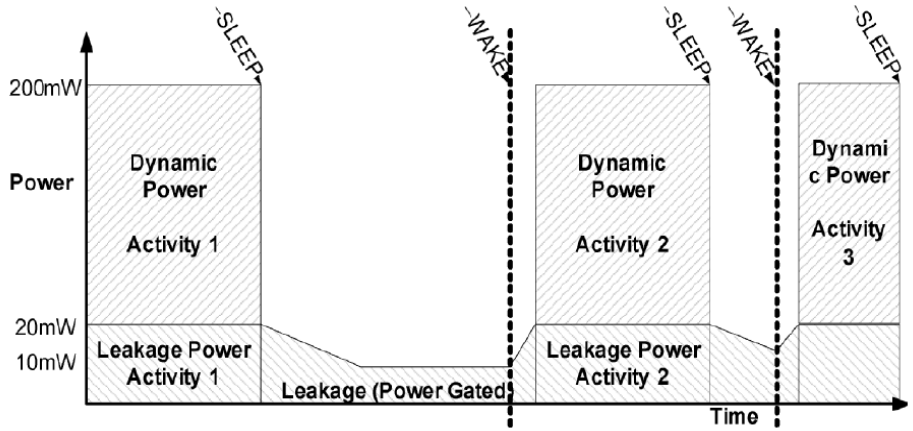
Power gating (cont.)



Clock gating



Power gating (ideal)

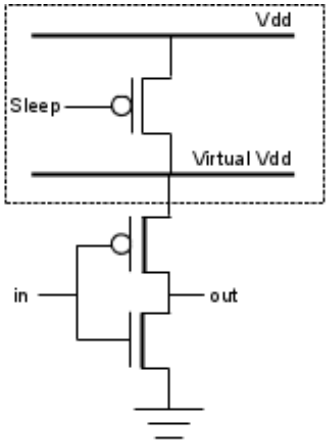


Power gating (more realistic)

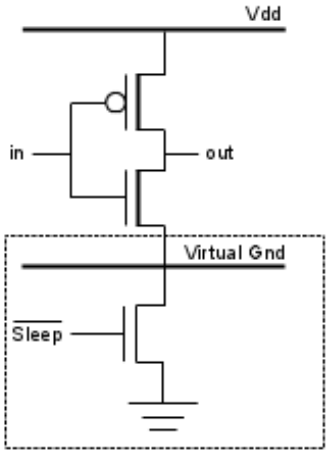
- Possible power saving is higher for power gating than clock gating
- Power gating requires time to drain/load capacitances of gated circuit!

(src: [Keating])

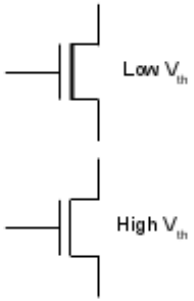
Power Gating (cont'd)



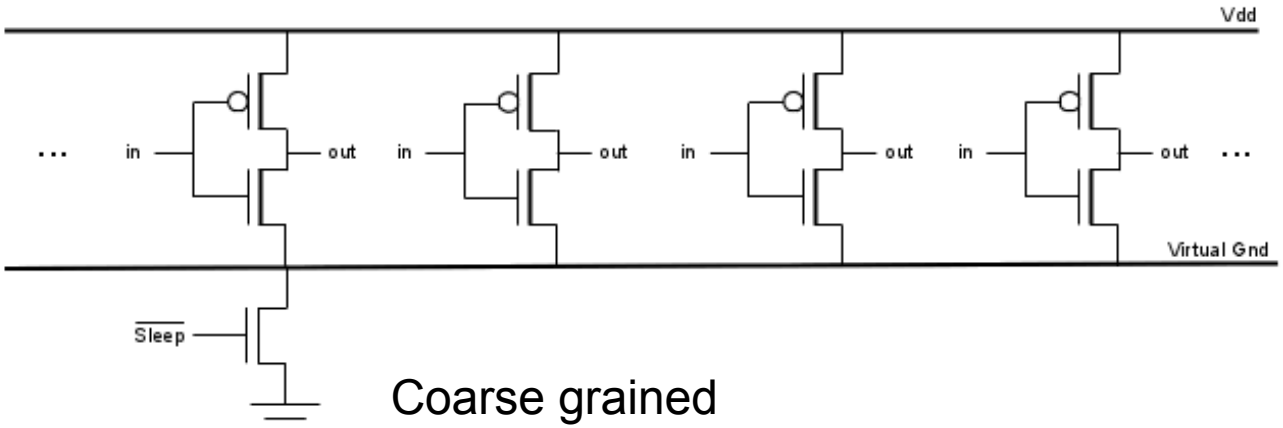
Header: V_{dd} is pulled to Gnd



Footer: Gnd is pulled to V_{dd}



Gating transistors are High V_{th} to limit leakage in gated mode



Coarse grained

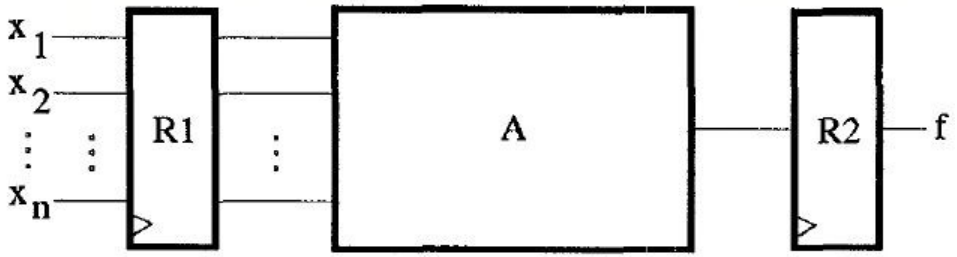
Power Gating (cont'd)

- Highest time requirement is loading circuit capacitance
  - Footer: most delay is at start of power gating ("switching off")
  - Header: most delay is at end of power gating ("turning on")
- Trade-off between sleep transistor size and speed of gating
  - Larger transistors can load/drain capacitances much quicker
  - (but also consume more leakage themselves!)
- Dynamic power is consumed when turning on/off power gating
  - If power gated interval too low may result in increase in power consumption
  - Break-even time can be determined.

Power savings through Pre-Computation

Power savings through pre-computation

original circuit



(Src: [Devadas])

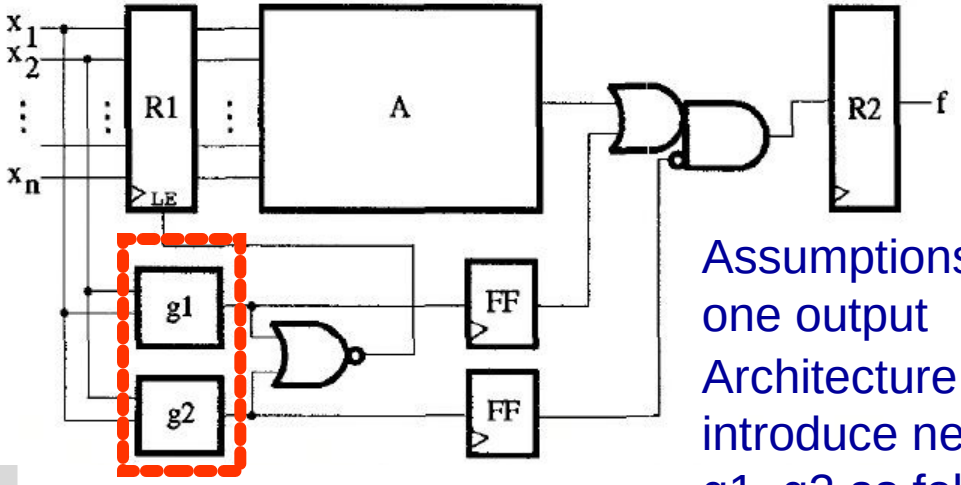
Basic idea:

Pre-compute (i.e. predict) output of a circuit one cycle ahead with additional logic and then switch off original logic

What is the rationale?

For a majority of input values, the output might be computed with very simple logic but in order to cover all input, complex logic is necessary

Circuit with pre-computation logic



Assumptions: 'A' represents combinational logic; has one output

Architecture:
introduce new functions g1, g2 as follows

$$g_1 = 1 \Rightarrow f = 1$$
$$g_2 = 1 \Rightarrow f = 0$$

Tasks of g1, g2: a) pre-compute, b) switch off original circuit in certain cases when prediction is possible

Note: a) g1, g2 only cover a subset of all x1,…,xn (desirable: a large coverage) b) imposes overhead in form of power area and probably performance

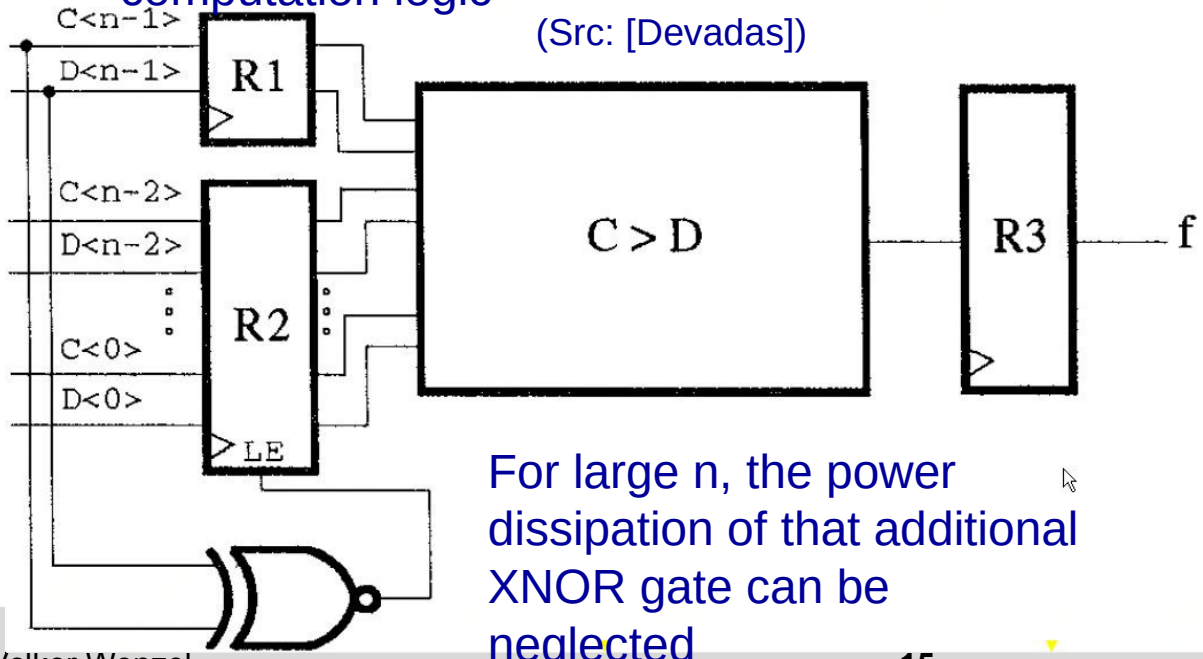
Power savings through pre-computation

- Example: a comparator of two n-bit values C and D that results in '1' if C > D

In that case, g1 can be defined to test the MSB:
Accordingly, g1 is defined as:
If g1=1 => C>D; g2=1 => D>C
So, XNOR needs to be computed for the pre-computation logic

$$g_1 = C\langle n-1\rangle \cdot \overline{D\langle n-1\rangle}$$
$$g_2 = \overline{C\langle n-1\rangle} \cdot D\langle n-1\rangle$$
$$\overline{g_1 + g_2} = C\langle n-1\rangle \otimes D\langle n-1\rangle$$

(Src: [Devadas])



For large n, the power dissipation of that additional XNOR gate can be neglected

Note: assuming a uniform probability for the inputs, the probability that XNOR results to '1' is 50% !
If the bit MSB-1 is also tested, the probability is 75%!

=> With little effort the circuit can be predicted and power can be saved by switching off (gating) the original circuit

Managing Power through Scheduling

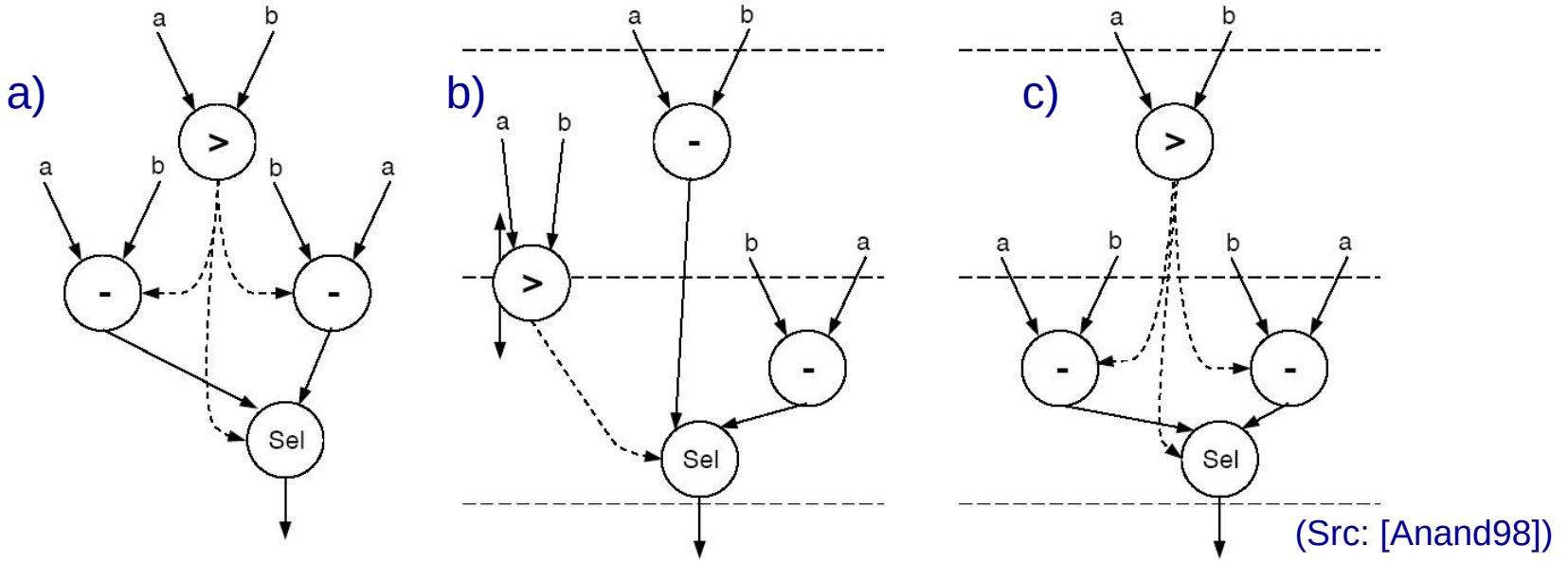
Managing power through scheduling

- Recall: “pre-computation” is a shut-off technique based on clock cycle basis and is limited to the given structure of the logic
- Can power be managed at a higher level?
- Observation: it is common for performance optimization to compute all outcomes of a conditional operation **in parallel** with the condition evaluation itself

  The appropriate result will be chosen and the other one(s) will be discarded
- => this is ineffective in terms of power consumption
- Idea: power effective: enforce control dependencies between operations in CDFG and conditional operation such that they depend during scheduling

  May be accomplished by meeting performance constraints first and then optimize power consumption

Managing power through scheduling (cont'd)



(Src: [Anand98])

Example: expression |a-b|
Assumption: each op '-' and '>'
Takes one clock cycle and 'sel' operation may be chained with any other operation
Constraint: a schedule within 2 cycles

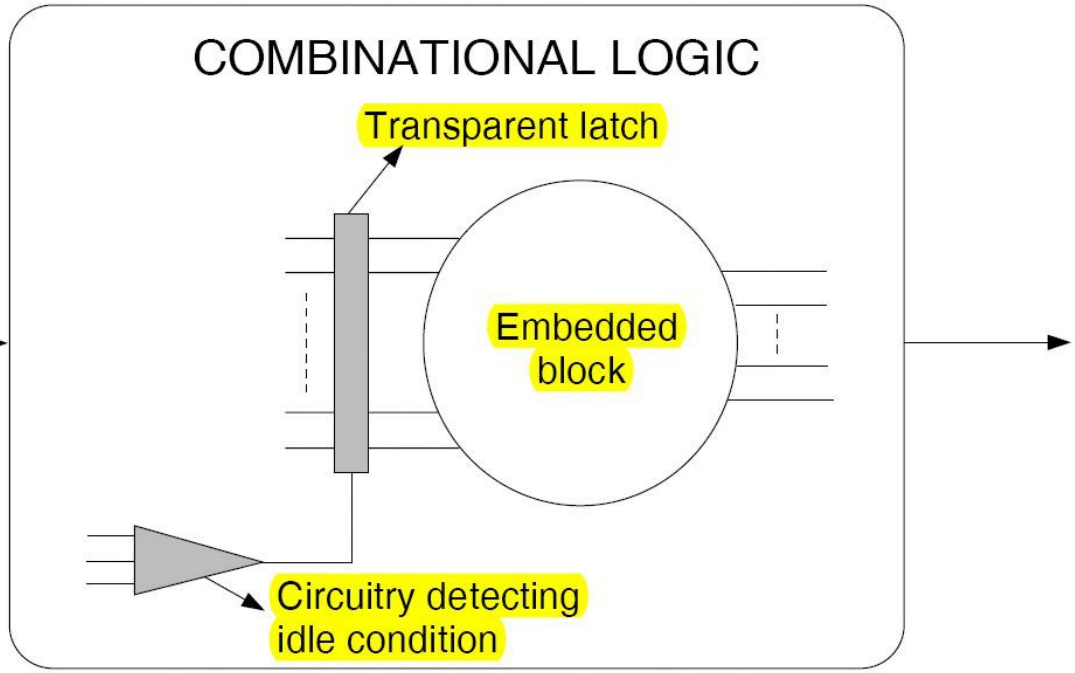
→ Two possible schedules b) and c)

Schedule b): ignores control dependencies; a-b and b-a are executed independently of '>'; There is a flexibility in scheduling '>'
Problem: from power point of view b) is inefficient: both a-b and b-a are always executed
Schedule c): a-b or b-a are activated exclusively due to outcome of '>'.
a-b, b-a may be assigned to same or to two different subtractors (latter case: one needs to be shut down)

Power savings through Operand Isolation

Power savings through operand isolation

- Note: previous techniques (pre-computation, gated clocks) are only applicable to blocks of combinational logic that are fed by registers
- Here: applicability to circuit blocks that are embedded within combinational logic
- Idea here:
  - Disable transitions at inputs of variables
  - Insert transparent latches at all inputs of embedded block
  - If block does not perform any useful operation: a) transparent latches at inputs are disabled, b) retain previous cycle's values
    - → avoids unnecessary power consumption



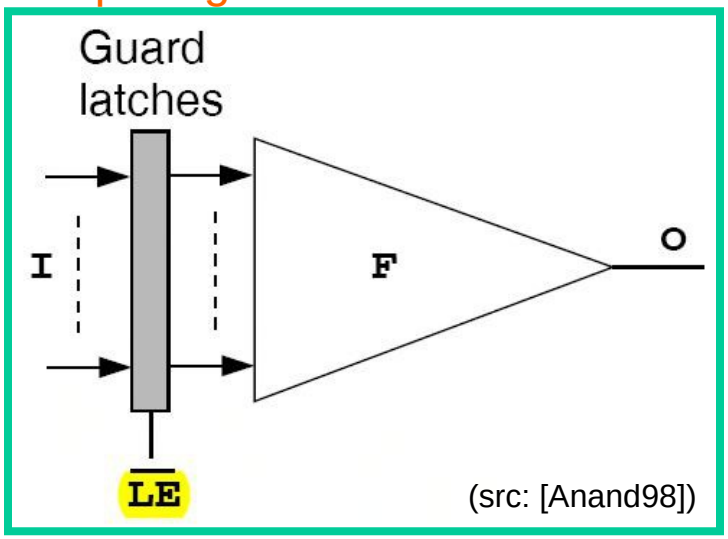
(src: [Anand98])

Power savings through operand isolation:
- guarded evaluation -

o – signal in a combinational circuit
F – logic that is computing o
I – set of inputs to F
ODC_o – observability don't care set with respect to o, i.e. set of primary input assignments to the entire circuit such that the value of o has no influence on the values at the primary outputs
LE – an arbitrary value of the existing circuit such that LE ⇒ ODC_o, i.e. $\overline{LE} + ODC_o \equiv 1$

Thus, when LE = 1, the value on o is not needed to compute the primary outputs.

pure guarded evaluation



(src: [Anand98])

$t_e(I)_{LE=1}$ earliest time at which any of the inputs in I can change its value when LE=1

$t_l(LE)_{LE=1}$ the latest time at which LE can stabilize to logic value '1'

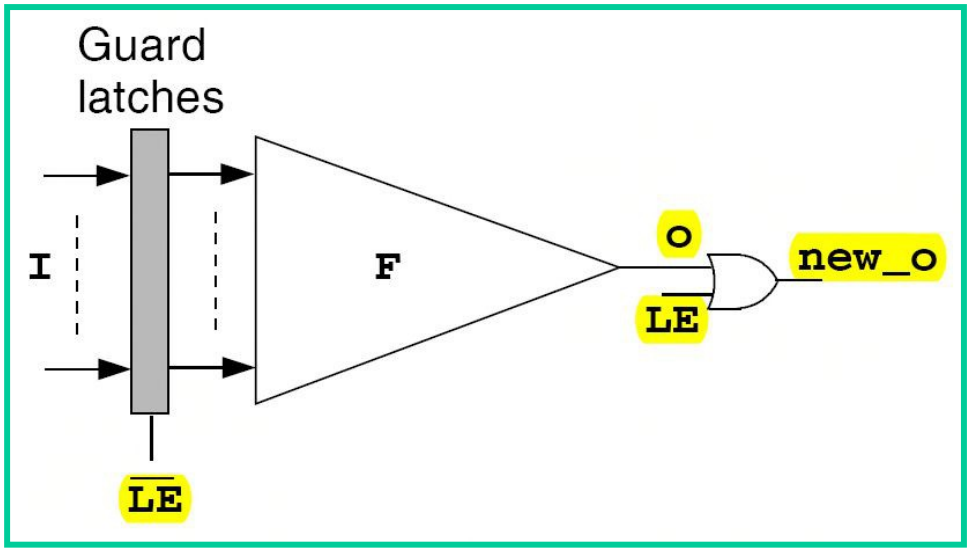
$t_l(LE)_{LE=1} < t_e(I)_{LE=1}$ LE can be used to control guard logic. Transparent latches need to be disabled in time, i.e., early enough to cut off transitions on any of the inputs in I (and such save power)

(src: [Tivari])

Power savings through operand isolation: - relaxed guarded evaluation (cont'd) -

Idea: use guarded evaluation, but: use a relaxed condition such that it becomes easier to find the shut-off condition (remember: signal LE must be available anywhere in the existing circuit):

$$\mathrm{LE} \Rightarrow (o + ODC_o) \Leftrightarrow \overline{\mathrm{LE}} + o + ODC_o \equiv 1$$



(src: [Anand98], [Tivari])

Timing condition: same as before at "pure guarded evaluation"

Two cases for LE=1 (note: for LE=0 circuit is functioning correctly anyway)

1. o is not needed to compute primary outputs (OK)
2. o is needed to compute primary output. Circuit may operate incorrectly. An 'OR' gate is needed (see figure)

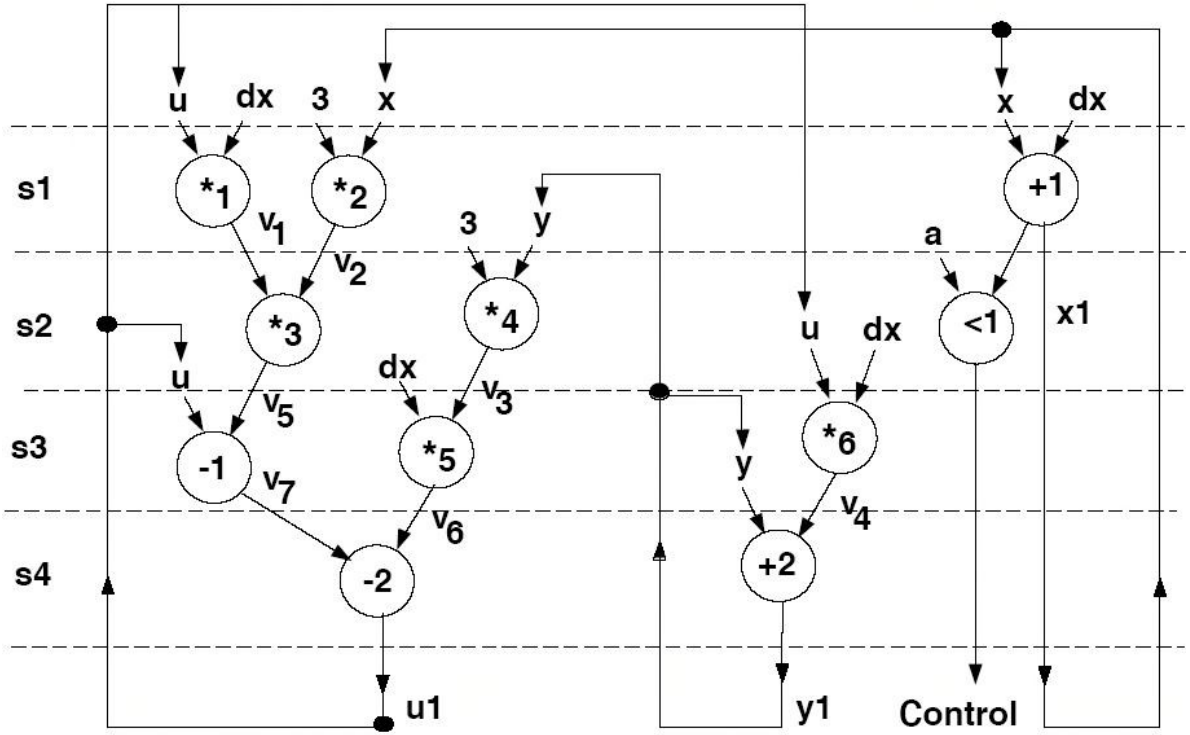
Power savings through operand isolation:
- guarded evaluation -

Comparing: pre-computation and guarded evaluation

- **Pre-computation** needs additional circuitry; **Guarded Evaluation** is derived from within the circuit
- **Pre-computation** may require re-synthesis to efficiently derive additional circuits whereas **Guarded Evaluation** leaves circuit as is (especially important in hand-optimized circuitry)

Power savings through operand isolation:
- application to high-level synthesis -

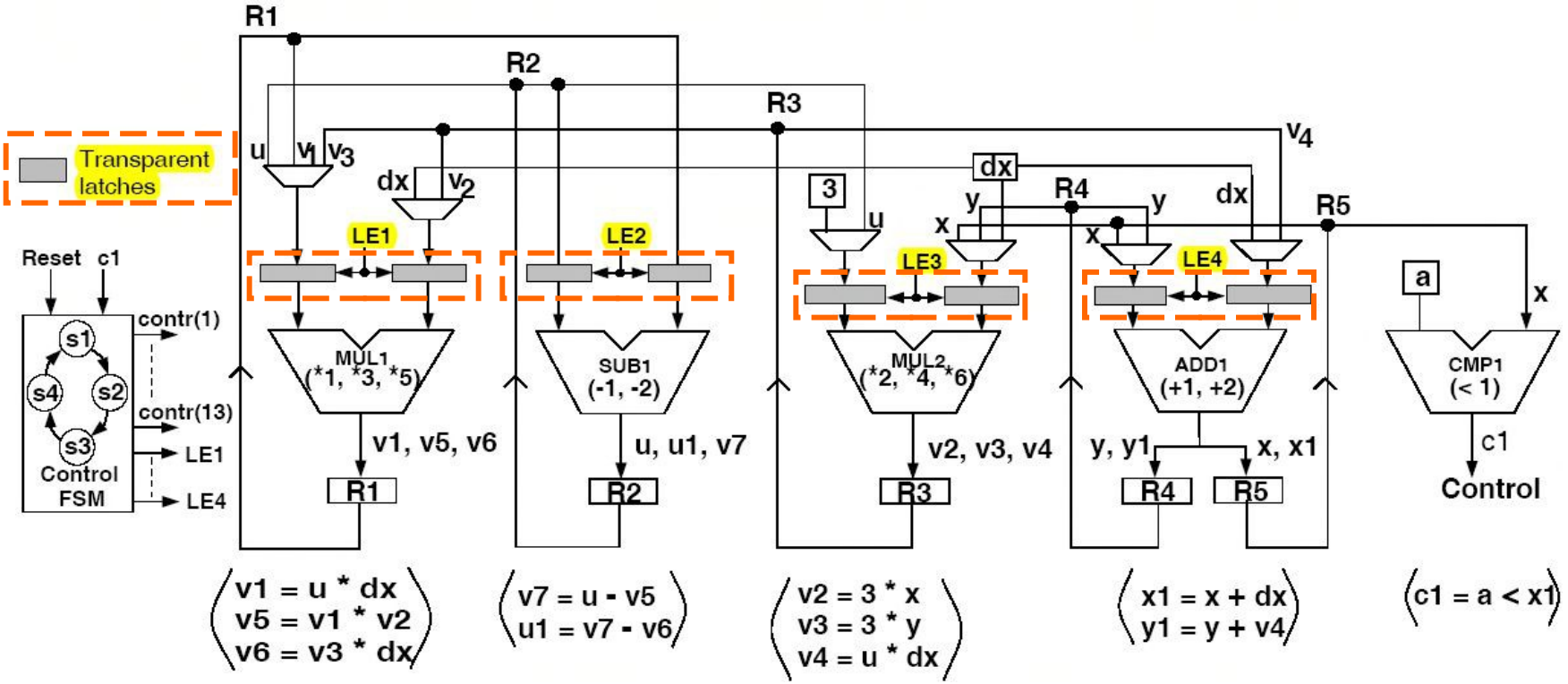
- Idea: apply operand isolation from logic level to high-level synthesis
- In fact: the conditions under which a resource (*e.g.* a functional unit FU) is not used are readily available from the scheduling and resource sharing!
- => idle cycles can be derived from the circuits (see next slide)



Scheduled example DFG

(src: [Anand98])

Power savings through operand isolation:
- application to high-level synthesis -



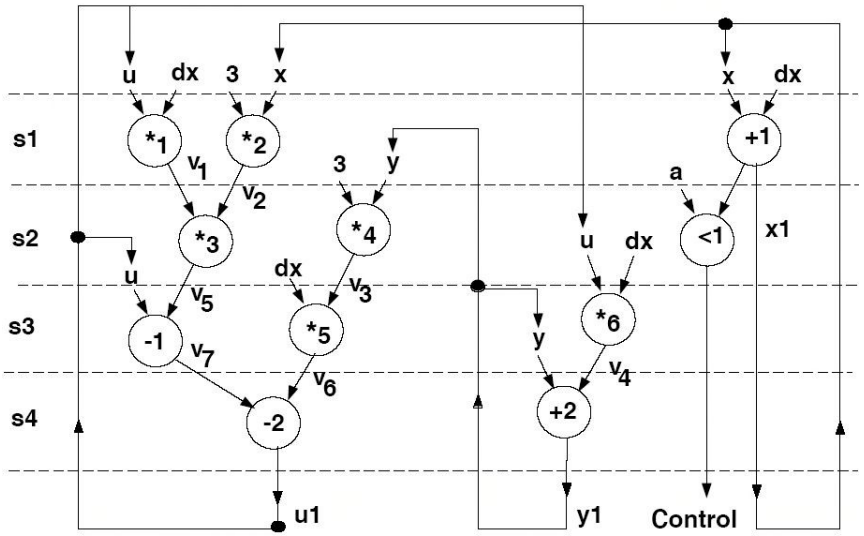
(src: [Anand98])

Power savings through operand isolation: - application to high-level synthesis (cont'd)

- Idle cycles of FUs
  - MUL1, MUL2: s4
  - ADD1 : s2, s3
  - SUB1 : s1,s2
  - CMP1 : s1, s3, s4
- Insert transparent latches at the FU's input to perform operand isolation
  - LE1 = LE3 = x4
  - LE3 = x1 + x2
  - LE4 = x2 + x3

Q: Why no latches at the inputs of CMP1?

A: Input values do not change in idle cycles



(src: [Anand98])

Power savings through operand isolation:
- application to high-level synthesis (cont'd) -

Possible disadvantages:

The isolation technique attempts to eliminate spurious activity at the inputs of embedded resources (e.g. functional units) by inserting transparent latches into the RTL implementation.

- incurring power and area overheads due to the addition of extra circuitry
- operand isolation also requires some delay constraints (the disabling transition at the transparent latch enable input should arrive before its data input can change).
- Satisfaction of the delay constraints may require the addition of extra circuit delay in the critical path, which may not be acceptable for high-performance designs.

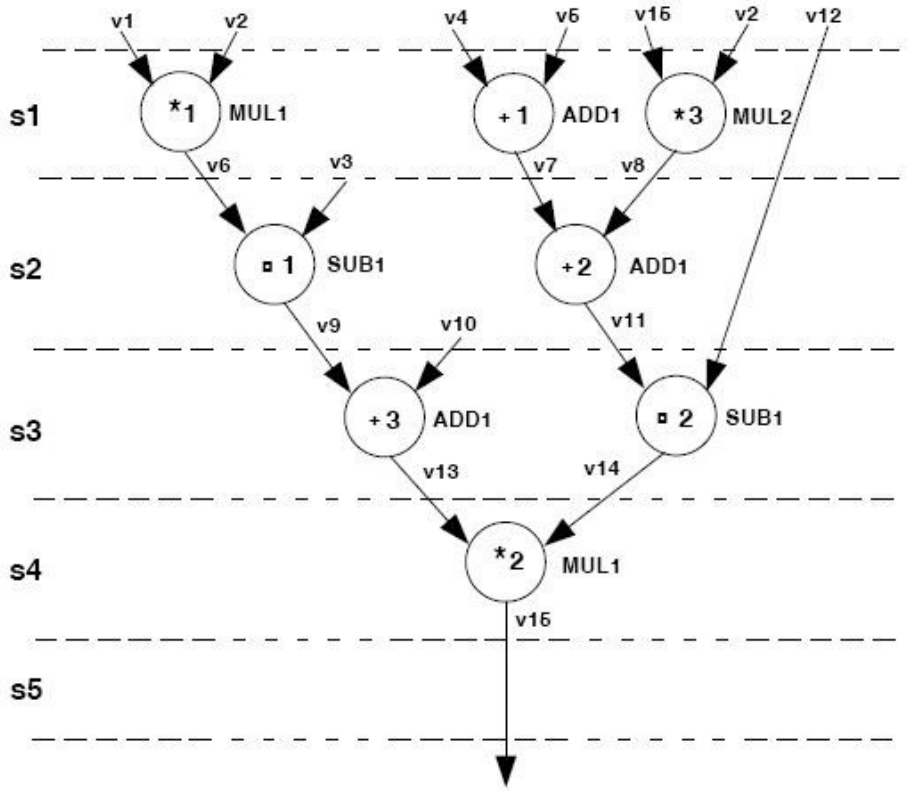
Power savings through constrainted register sharing

Power savings through constrained register sharing

- Idea: rather than applying transparent latches to an already scheduled DFG, can't the scheduling, mapping etc already take into consideration power shut-down techniques?
- Impact of variable assignment on power consumption:
  - two candidate assignments, Assignment 1 and Assignment 2, shown in Table (next slide).
  - Architectures obtained using these assignments were subject to:
    - logic synthesis optimizations, and placed and routed using a cell library.
    - The transistor-level netlists extracted from the layouts were simulated using a switch-level simulator with typical input traces to measure power.
  - For the circuit Design 1, synthesized from Assignment 1, the power consumption was 30.71mW, and for the circuit Design 2, synthesized from Assignment 2, the power consumption was 18.96mW !

Power savings through constrained register sharing (cont'd)

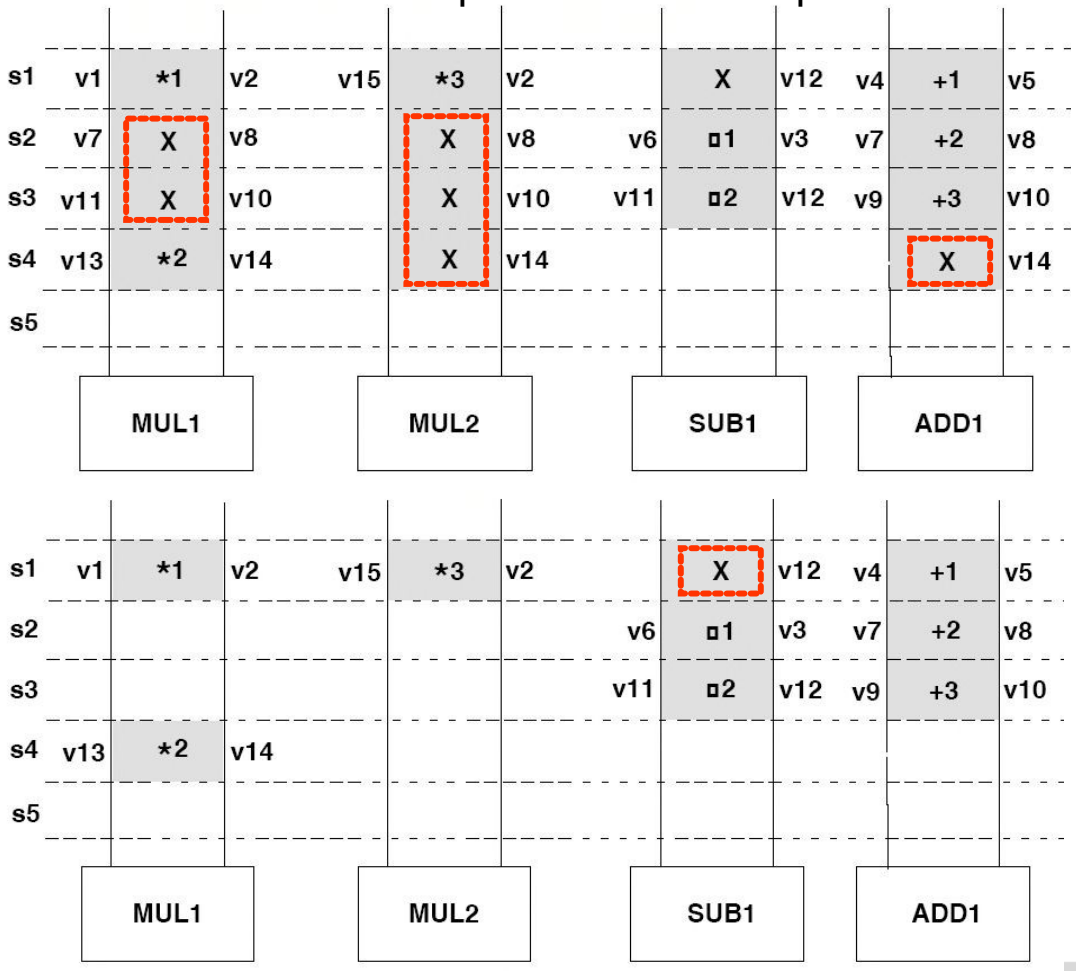
Example: DFG and two possible register assignments that differ significantly in power consumption



Register	Assignment 1	Assignment 2
R1	v1, v7, v11, v13	v1, v13
R2	v2, v8, v10, v14	v2
R3	v3, v5, v9	v4, v8, v10
R4	v4, v6	v5, v7, v9
R5	v12	v12
R6	v15	v3
R7	-	v6, v11
R8	-	v14
R9	-	v15

Power savings through constrained register sharing (cont'd)

Two distinct schedules: Shown: FU (in box), input variables left and right of operation, grey-shaded: variables at input of respective operation change value; spurious input transitions i.e. those that do not correspond to an DFG operations are marked with an 'X'.



Observation:

A functional unit that does not alter its input does not perform a spurious operation

Conclusion:

Constrained register sharing can save significant energy/power:

Upper case: 7 operations that do not correspond to a DFG operation

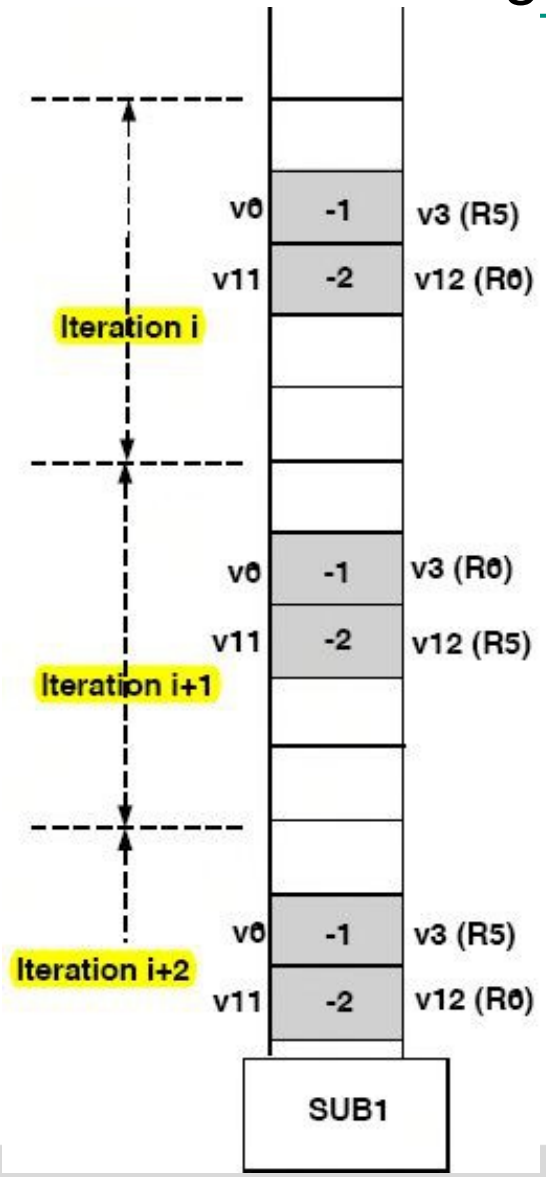
Lower case: only one such operation

Note:

a) Number of control steps is not increased

b) Number of HW resources (FUs) is still the same

Power savings through dynamic variable rebinding



Problem in previous example:

-There is still a spurious operation is control step s1 of each operation:

-MUX selects R5 (to which v12 is assigned) from control step s3 of each iteration to control step s1 of the next iteration

- v12 acquires a new value at step s1

Idea::

Combine dynamic variable rebinding with variable assignment to completely eliminate spurious operation

How::

-Need to preserve old (previous iteration) value of v12 at input of SUB1 until new value of v3 in current iteration is generated

- then, spurious operation can be eliminated

- swapping the variables assigned to registers R5 and R6 in alternate iterations

- result: see figure on the right

(src: [Anand98])

Managing power through the controller

- Main idea:
  - Find way to reduce power without having to spend large overhead in terms of hardware (like transparent latches, etc.)
- Rather: Re-design existing control logic in order to reconfigure the multiplexer networks and functional units in the data path
- Might not completely eliminate activity but is low-cost
- best suited to control flow intensive designs:

Managing power through the controller

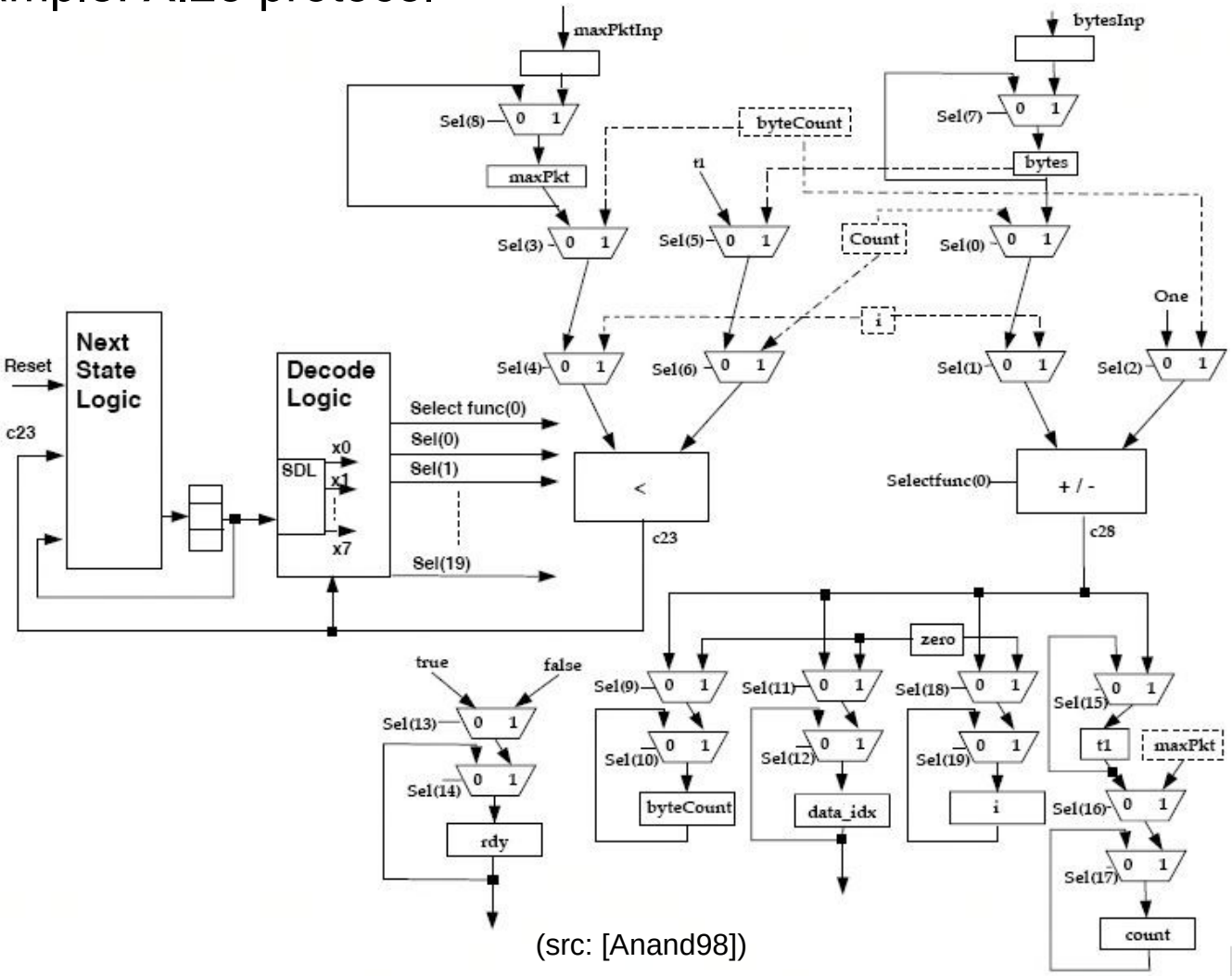
- Power consumption is dominated by an abundance of smaller components like multiplexers, while functional units may account for a small part of the total power . The power overheads due to the insertion of transparent latches is comparable to the power savings obtained when power management is applied to sub-circuits such as multiplexer networks.

- The signals that detect idle conditions for various sub-circuits are typically late-arriving (for example, due to the presence of nested conditionals within each controller state, the idle conditions may depend on outputs of comparators from the data path). As a result, the timing constraints which must be imposed in order to apply conventional power management techniques (the enable signal to the transparent latches must settle before its data inputs can change) are often not met.

(src: [Anand98])

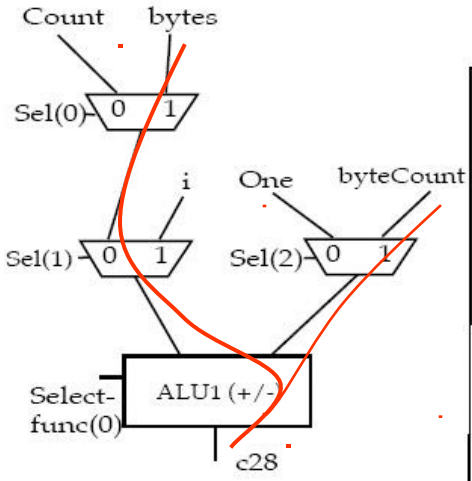
Managing power through the controller (cont'd)

- Example: X.25 protocol

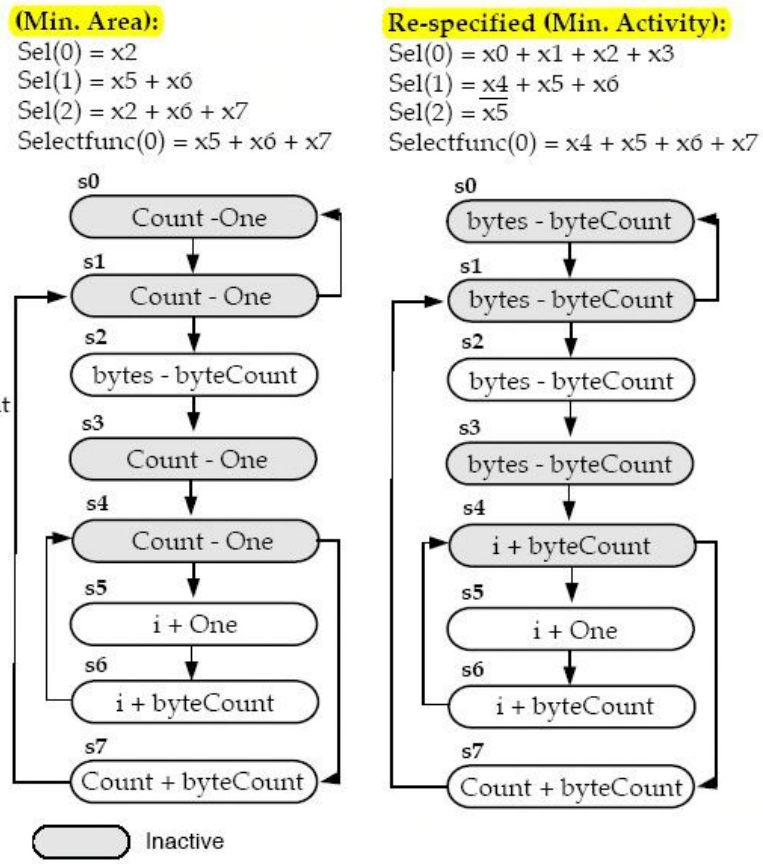


(src: [Anand98])

Managing power through the controller: X.25 example 1



(src: [Anand98])



Shown:

- left: a part of the data path
- middle/right: a) logic expression for control signals (x_i=1 => controller is in state s_i); b) activity graphs of ALU (state transitions with actions involved; ex: sel(0) sel(1), sel(2), SelectFunc(0) are 1, 0, 1, 0, respectively => "bytes-byteCount" is performed

Observation:

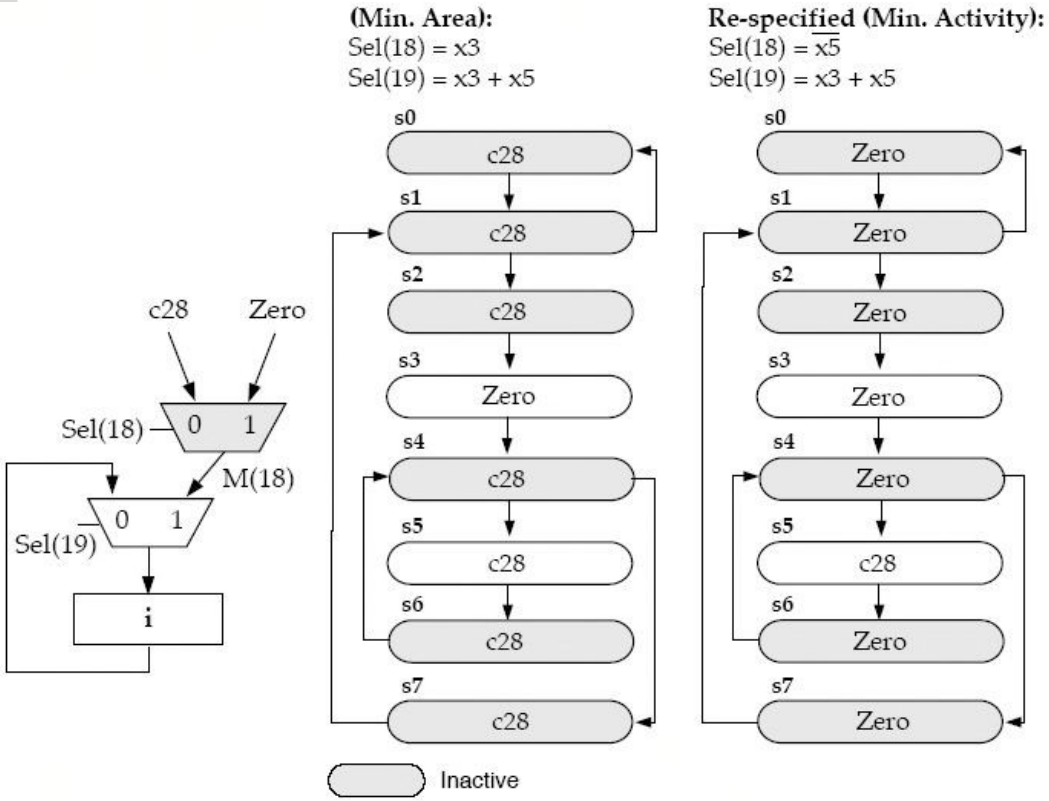
- some states are actually idle states (gray shaded). This can be found out through scheduling info from HL synthesis.

Idea:

- re-specify idle states such that switching is minimized.
- Example: state transition s6->s4 : same signals are on muxes such that same operands stay stable. Since they do not change => no switching => no unnecessary power consumption

Conclusion: re-specifying control signals can lead to power savings (without changing anything else)

Managing power through the controller: X.25 example 2



Shown:

- Different part of the X.25 designs: register that stores variable I and muxes that feed it through signals sel(18), M(18), sel(19)
- Same convention as before apply (gray shaded states are inactive states as far as that respective part of the design is concerned)

Idea:

- Can reduced activity in the mux tree lead to reduced power consumption?

Idea:

- Consider s7->s1: “count+byteCount” -> “bytes-byteCount). Since operation changes, also operand c28 changes (c28 is output of ALU and feed the mux tree)

Solution:

- re-specifying the signals prevents from propagating this variable into the shown mux tree (see right state diagram)

Managing power through the controller (cont'd)

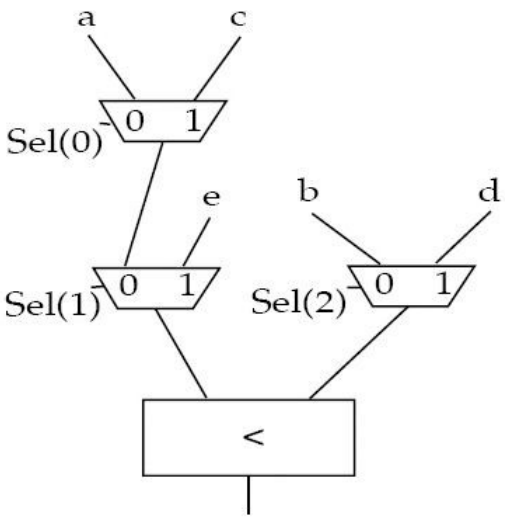
- Re-labeling activity graphs:

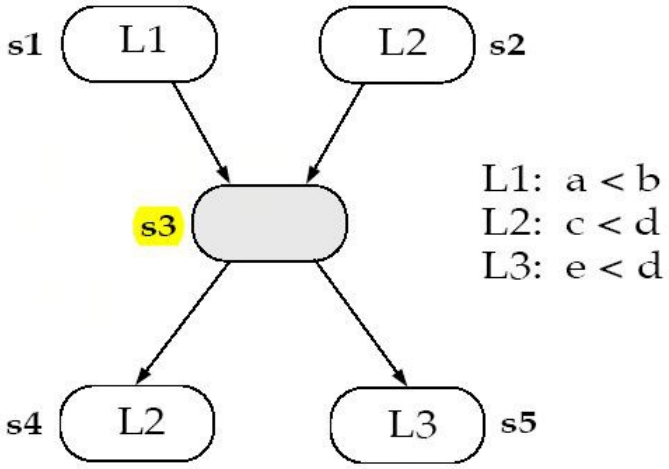
  How to label an idle vertex in an activity graph?

  - Different incoming and outgoing transitions into the idle state have different execution probabilities

  - The values of data operands fed to the mux trees may themselves change => Only selecting the same operand does not ensure that switching activity is minimized

Managing power through the controller: formalizing the problem





Shown:
- A part of a data path and a part of the activity graph for the "<" operator
- s3 is an idle state
- shown are all incoming and outgoing arcs to/from s3

Goal:
- using s3 for one of the labels L1, L2, L3 such that activity at the input of the "<" is minimized

Some conventions:
- P(si - > sj) - probability of the controller state transition from si to sj
- AM $_{si->sj}$ - activity matrix stores the cost of (average bit transitions for respective state transition). Has only entry in row/column if the respective transition is actually in the state transition graph
- Nodes in the state transition graph are to be re-labeled while minimizing labeling costs:

$$Labeling\ Cost = \sum_{allsi \rightarrow sj} AM_{si \rightarrow sj}[L(si), L(sj)].P(si \rightarrow sj)$$

$$AM_{s1\rightarrow s3}[L1, L^*].P(s1 \rightarrow s3) + AM_{s2\rightarrow s3}[L2, L^*].P(s2 \rightarrow s3) + AM_{s3\rightarrow s4}[L^*, L2].P(s3 \rightarrow s4) + AM_{s3\rightarrow s5}[L^*, L3].P(s3 \rightarrow s5)$$

Goal: find an L* such that cost function is minimized => best re-labeling

Conclusion: Hardware power

- HW power sources:
  - Data path
  - Control path
  - Clock tree
- Optimization strategies:
  - Operator scheduling for low power
  - Hardware power management (clock gating)
  - Re-labeling of controller
- Very often there is a tradeoff
  - reduced power may lead to:
    - more logic
    - more complex design
    - reduced performance
    - …

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