Karlsruhe Institute of Technology

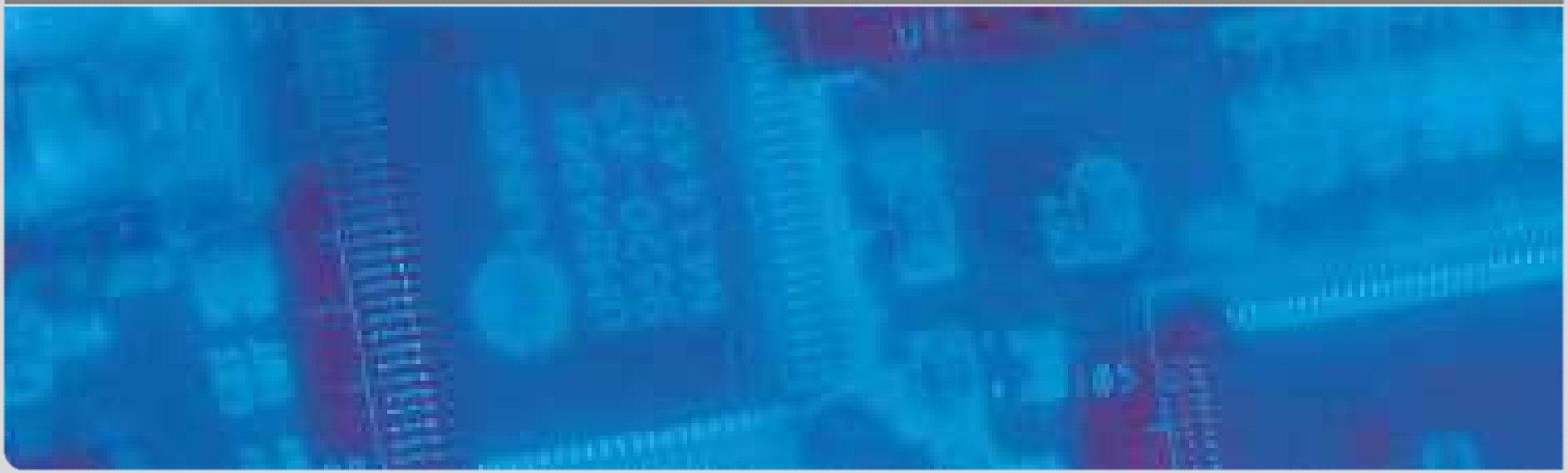
# Lower Power Design

*Lecture 5:* Hardware Power Optimization and Estimation 2

**Anuj Pathania on behalf of Prof. Dr. Jörg Henkel**

**Summer Semester 2017**

CES – Chair for Embedded Systems



# Organizational Issues

- Slides available for download -
  - http://cesweb.itec.kit.edu/teaching/LPD/s17_slides/
  - Username: student
  - Password: CES-Student
- Homework
  - Read a relevant scientific paper.
  - Discussion next class.
- Oral Exam
  - Make appointment with KIT CES secretary 6-8 weeks in advance.
  - Exam will be in English (or German if told in advance).
  - More information: http://ces.itec.kit.edu/972.php

# Lectures

- 27.04.2017 – ~~Lecture 0: Introduction~~
- 04.05.2017 – ~~Lecture 1: Energy Sources~~
- 11.05.2017 – ~~Lecture 2: Battery Modelling Part 1~~
- 18.05.2017 – ~~Lecture 3: Battery Modelling Part 2~~
- 25.05.2017 – Ascension Day (Holiday)
- 01.06.2017 – ~~Hardware Power Optimization and Estimation 1~~
- 08.06.2017 – Hardware Power Optimization and Estimation 2
- 15.06.2017 – Corpus Christi (Holiday)
- 22.06.2017 – Hardware Power Optimization and Estimation 3
- 29.06.2017 – TBA
- 06.07.2017 – TBA
- 13.07.2017 – TBA
- 20.07.2017 – TBA
- 27.07.2017 – TBA

# Overview for Today

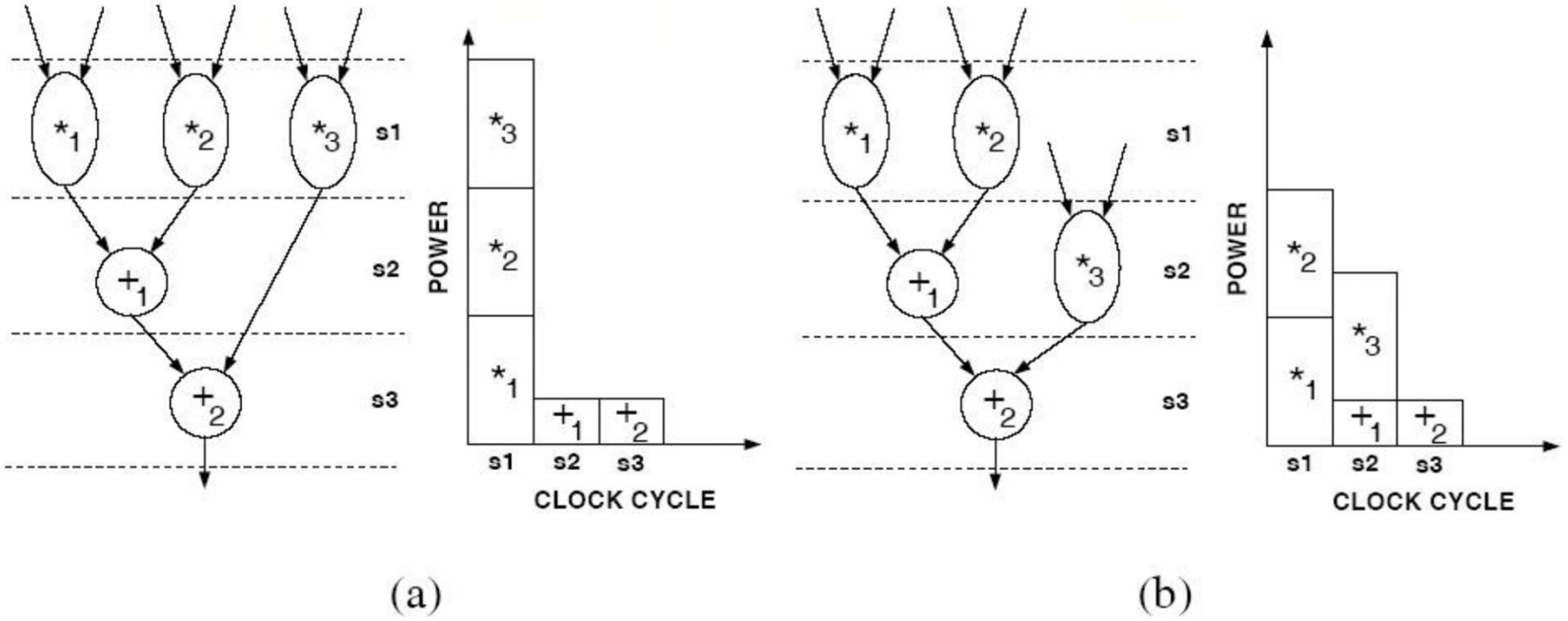
- Recap: Module Selection.
- Peak Power.
- Resource Sharing.
- DFG Restructuring.
- Glitch Power Reduction.
- Clock Gating.

# Recap: Module Selection

- Module Selection
  - Process of mapping operations from the DFG to component templates of RTL library.
  - Optimize power by choosing the most suitable component from template library.
- Example: “+” Operation
  - Implementation alternatives:
    - Ripple-Carry Adder (Slow, but more efficient in the switched capacitance)
    - Carry-Lookahead Adder (faster, but less efficient in switched capacitance)
    - …
- Similar tradeoffs exist for other operations.

# Peak Power

- Example: two possible schedules of a given DFG
  - 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.



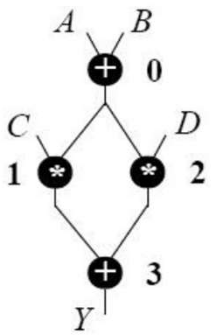

Source: Raghunathan [2012]

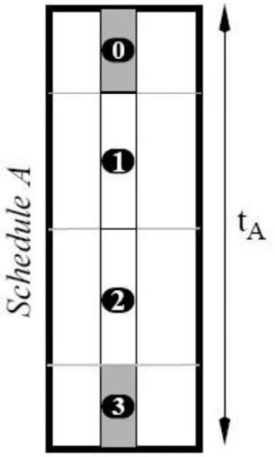
# Peak Power

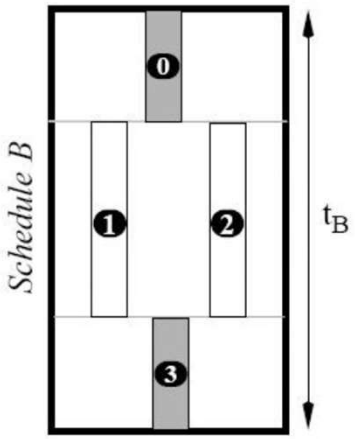
- Assumption: 1,2 and 0,3 may pair-wise share a module.
- Resulting delays $t_A$, $t_B$: $t_A = 2 \times t_{ADD} + 2 \times t_{MUL}$ $t_B = 2 \times t_{ADD} + t_{MUL}$
- Resulting average power consumptions:

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

  - Example: $t_a$ = 1300 ns; $t_b$ =660 ns; $P_a$ = 28.6 mW, $P_b$ = 56.4mW
  - Note: average power improved considerably, but energy is approx. the same!



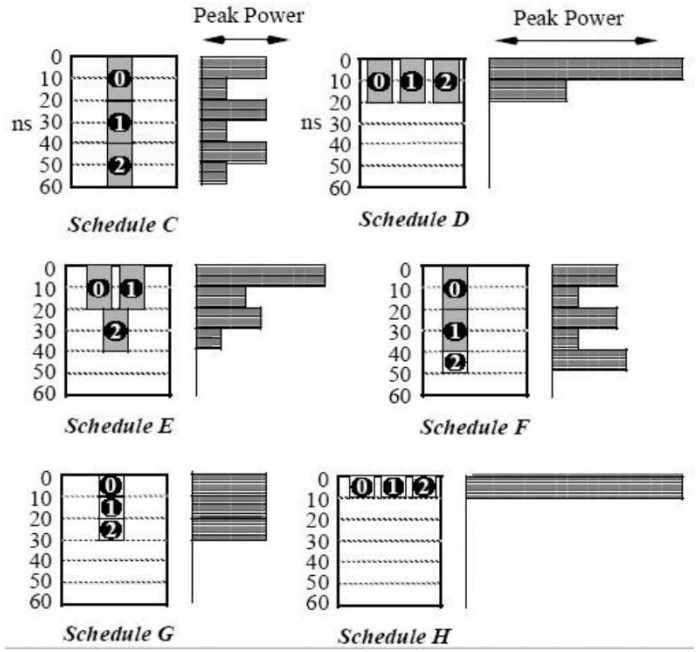




| | | 3.3V | 5V |
|---|---|---|---|
| RCA | Total Delay (ns) | 30.0 | 20.0 |
| | Power (mW) | 5.4 | 22.7 |
| CLA | Total Delay (ns) | 20.0 | 10.0 |
| | Power (mW) | 10.5 | 37.3 |
| QBS | Total Delay (ns) | 640.0 | 640.0 |
| | Power (mW) | 11.00 | 28.5 |
| BOOTH | Total Delay (ns) | 320.0 | 160.0 |
| | 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 |

Source: Martin [1995]

# Peak Power 2

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




| 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 |

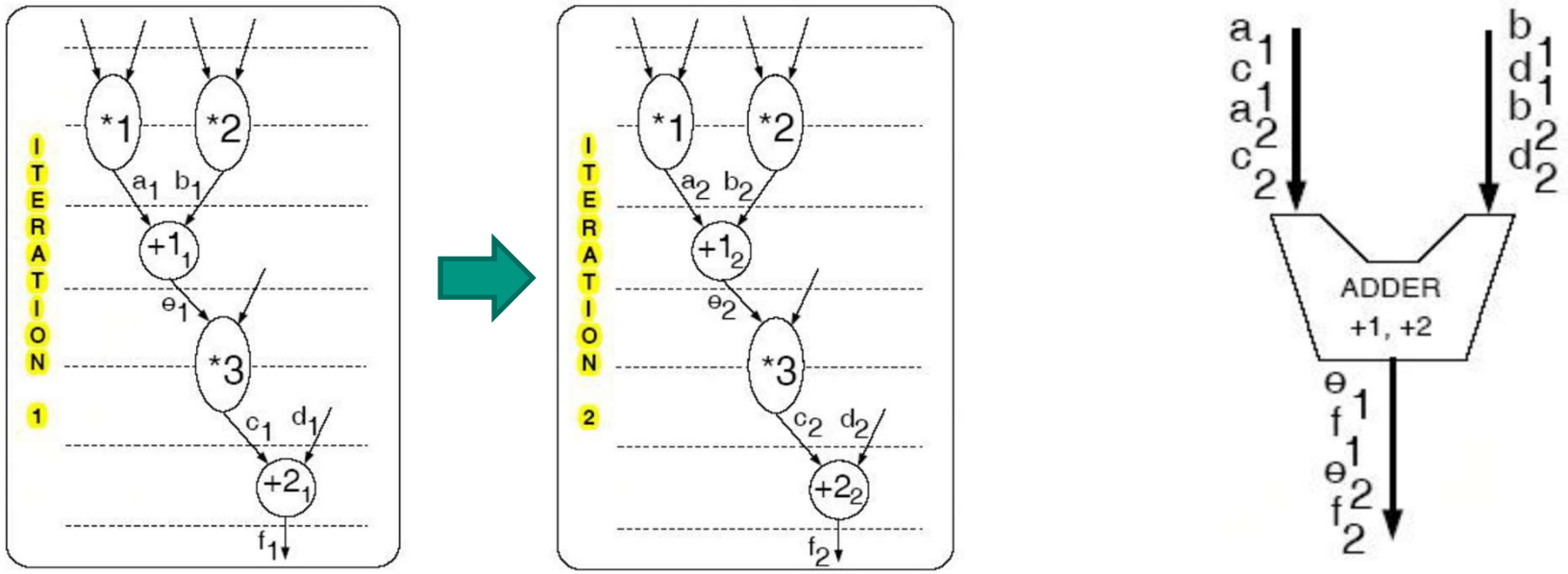
Source: Martin [1995]

# Resource Sharing for Low Power

- Resource Sharing is
  - Mapping OPs and variables in DFG 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 2

- Exploiting Signal Correlation –
  - 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?




Source: Raghunathan [2012]

# Resource Sharing for Low Power 3

- 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 4

- 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.
- 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 |

Source: Raghunathan [2012]

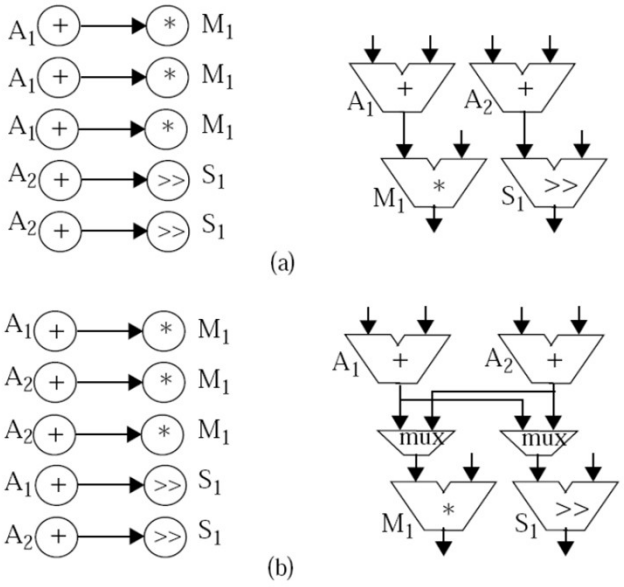
# Resource Sharing for Low Power 5

- 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 .

Source: Mehra [2012]

# Resource Sharing for Low Power 6

- 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 interconnects → more switched capacitance.
  - Overhead from multiplexers (and probably drivers) leads to more switching activity.
  - Conclusion: b) does not preserve regularity, a) does.



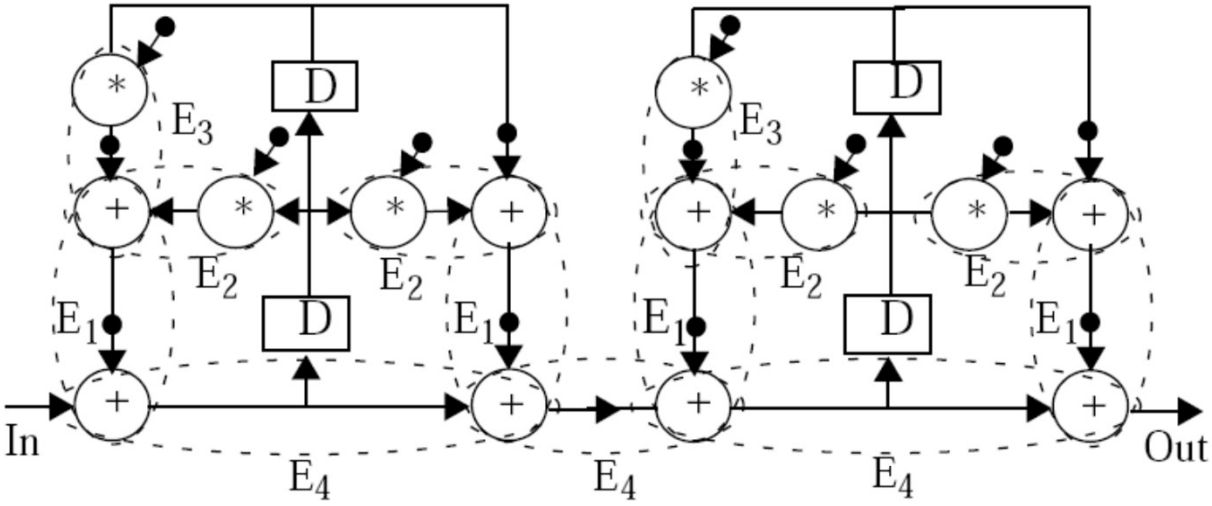

Source: Mehra [1997]

# Resource Sharing for Low Power 7

- 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.
- Disadvantages
  - may require more hardware units since sufficient E-templates need to be provided -> under circumstances power savings come at cost of more hardware.

Source: Mehra [1997]

# Resource Sharing for Low Power 8




| 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 |

Source: Mehra [1997]

# Resource Sharing for Low Power 9

- Example: E-template based scheduling.

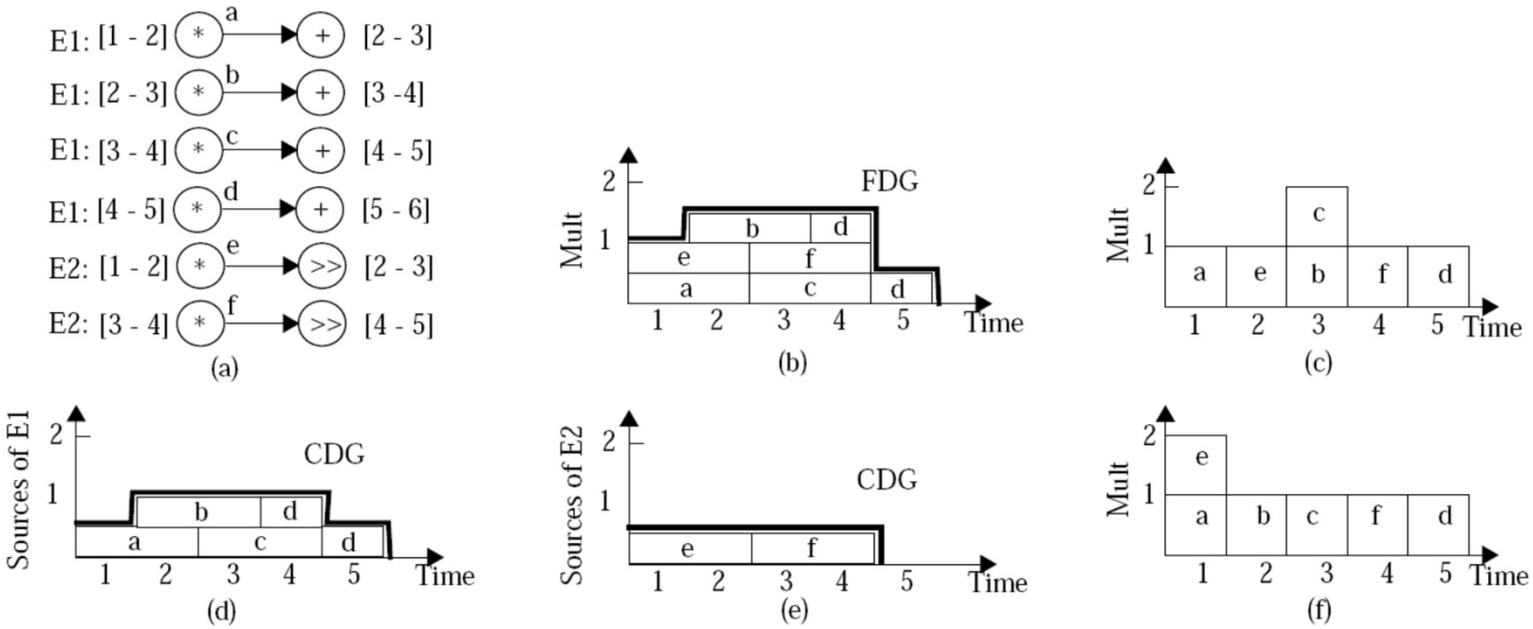



Figure 3. The effect of using connection distribution graphs, (a) Instances of two E-templates with their ASAP and ALAP times, (b) Initial FDG for multiply operations, (c) Final distribution graph using only FDGs, (d, e) Initial source CDGs of the two E-templates, (f) Final distribution graphs using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

Source: Mehra [1997]

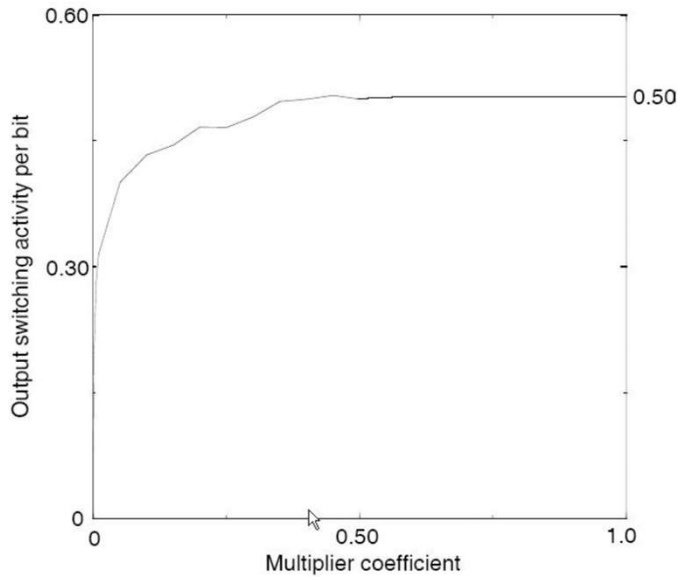
# DFG Restructuring for LP

- 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 2

- 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:

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

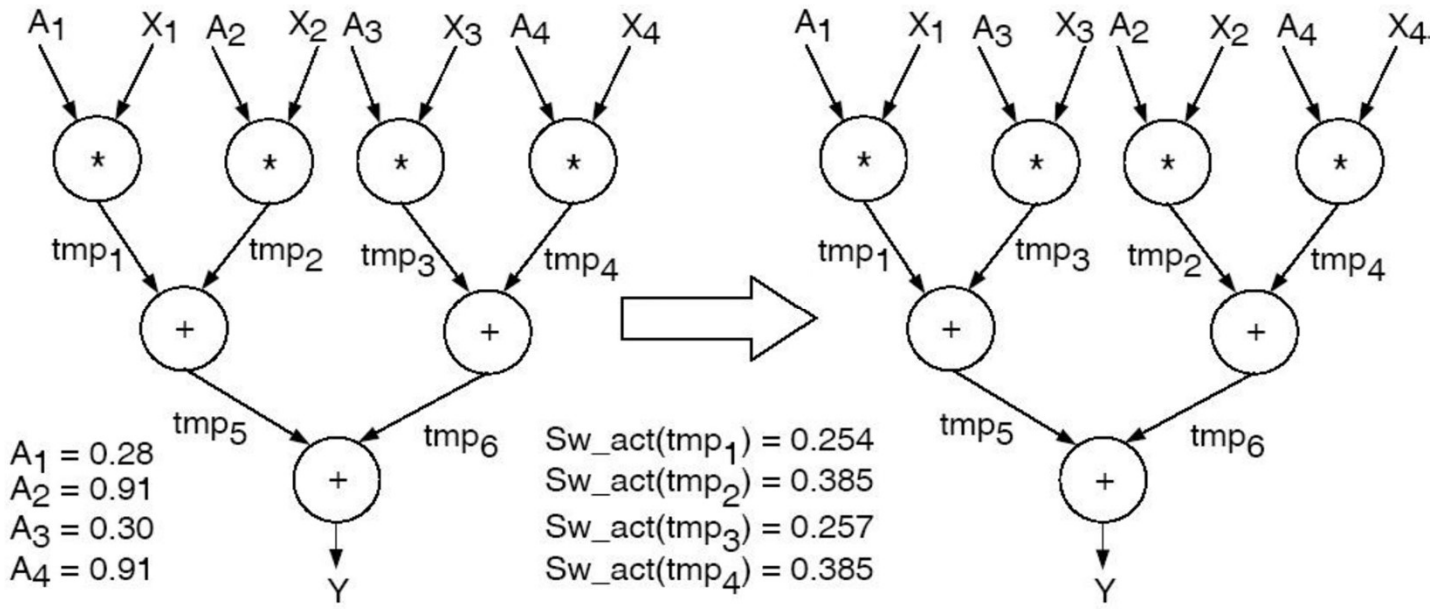


# DFG Restructuring for LP 3

- Example: linear time-invariant signal processing system (e.g. IIR filter).
- Left Figure $Y = \sum_{i=1}^{n} A_i.X_i$ Right Figure

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

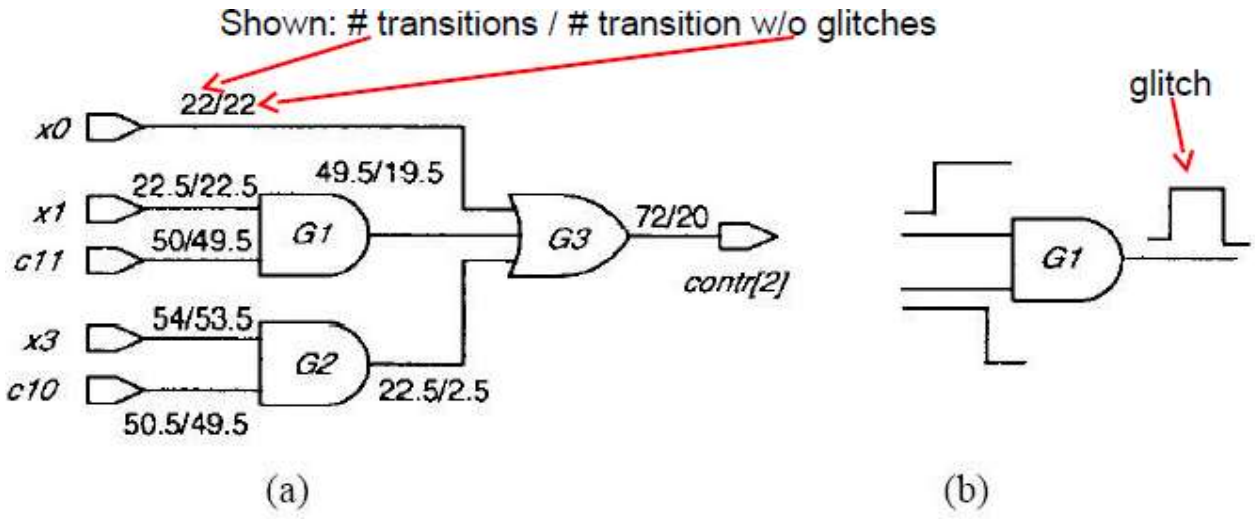
$$
\begin{aligned}
Sw\_act(tmp_5) &= \max(Sw\_act(tmp_1), Sw\_act(tmp_3)) = 0.257 \\
Sw\_act(tmp_6) &= \max(Sw\_act(tmp_2), Sw\_act(tmp_4)) = 0.385 \\
Sw\_act(Y) &= \max(Sw\_act(tmp_5), Sw\_act(tmp_6)) = 0.385
\end{aligned}
$$




Source: Raghunathan [2012]

# Glitch Power Reduction

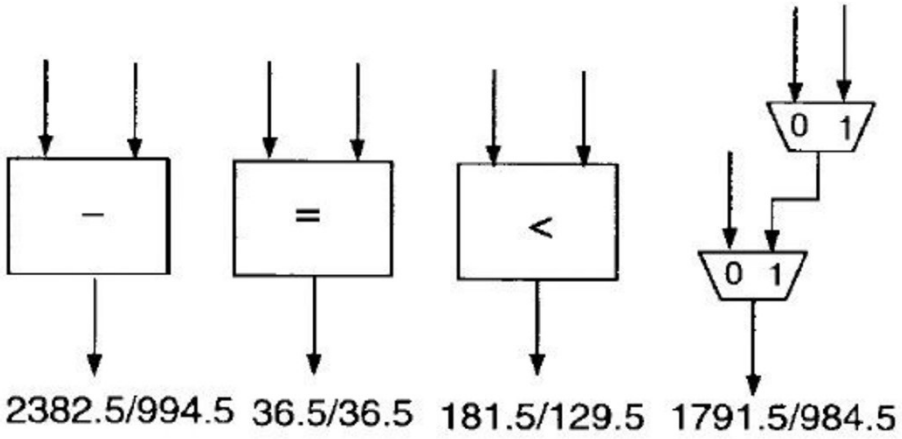
- What is “glitch power”?
  - Power consumption that is related to hazards i.e. temporary values at the input/output of gates that cannot be explained when considering a truth table only. It is due to different propagation delays in combinatorial paths.
  - Analysis of the below example unveiled:
    - 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.




Source: Raghunathan [1999]

# Glitch Power Reduction

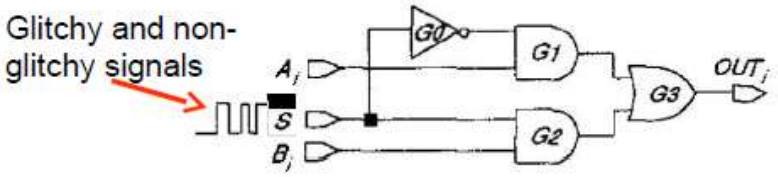
- In general, glitches are generated at the control signals due to the simultaneous presence of the following two conditions. 1) **Functional**: Correlation between rising and falling transitions at two or more signals that feed a gate. 2) **Temporal**: The controlling to non-controlling transition arrives 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.



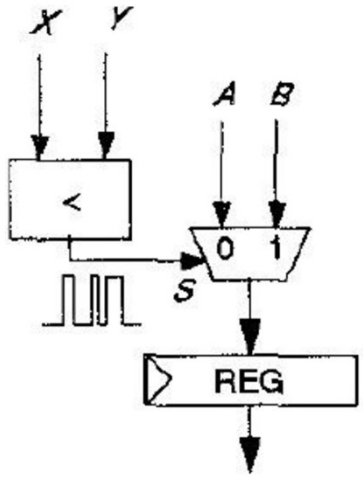

Source: Raghunathan [1999]

# Glitch Power Reduction

- 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.




| 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 |




- 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.

Source: Raghunathan [1999]

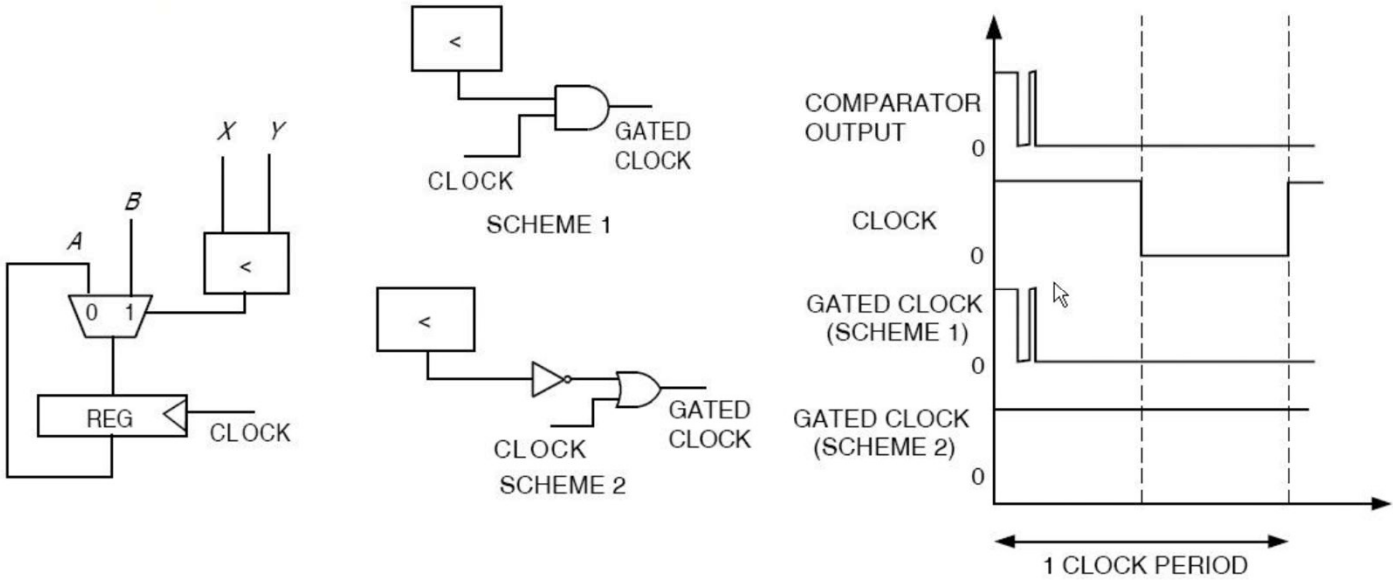
# Clock Gating

- 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.

Source: Raghunathan [1998]

# Clock Gating 2

- 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’).




Source: Raghunathan [1998]

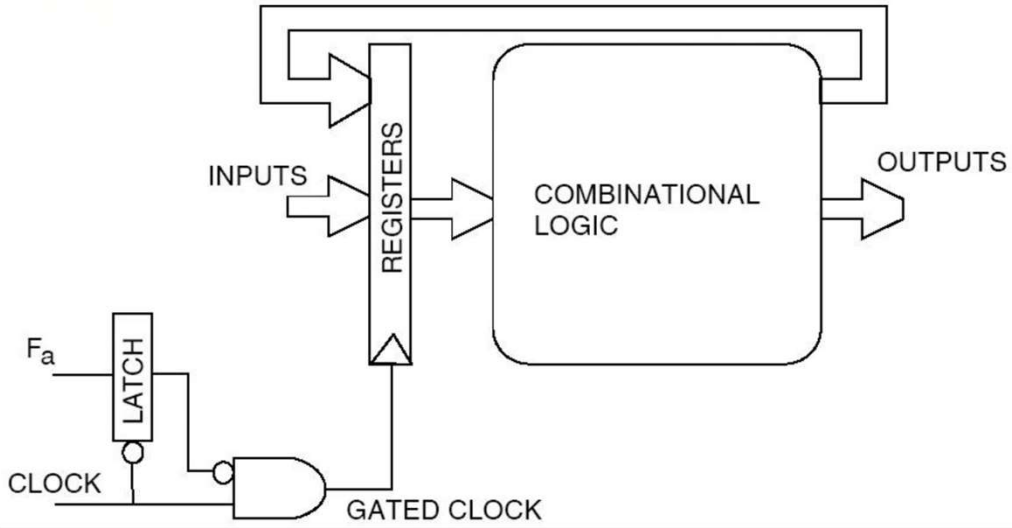
# Clock Gating 3

- 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.

Source: Raghunathan [1998]

# Clock Gating 4

- 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.
- Idea: automated gated-clock synthesis (architecture, see below).
  - 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,




Source: Raghunathan [1998]

# Clock Gating 5

- 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.
- Given: a Moore FSM $(PI, PO, S, s0, \delta, \lambda)$
  - (set of inputs, set of outputs, set of states, initial states, next-state function, output function).
  - State s_i with self-loop function

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

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 $s_i$

captures the set of input conditions under which the self-loop of state $s_i$ is traversed.

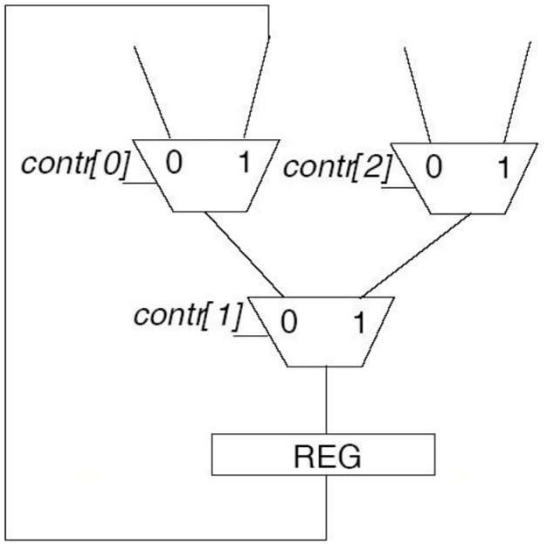
  - Activation Function (weight can be complex):

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

Source: Raghunathan [1998]

# Clock Gating 6 (For Data Paths)

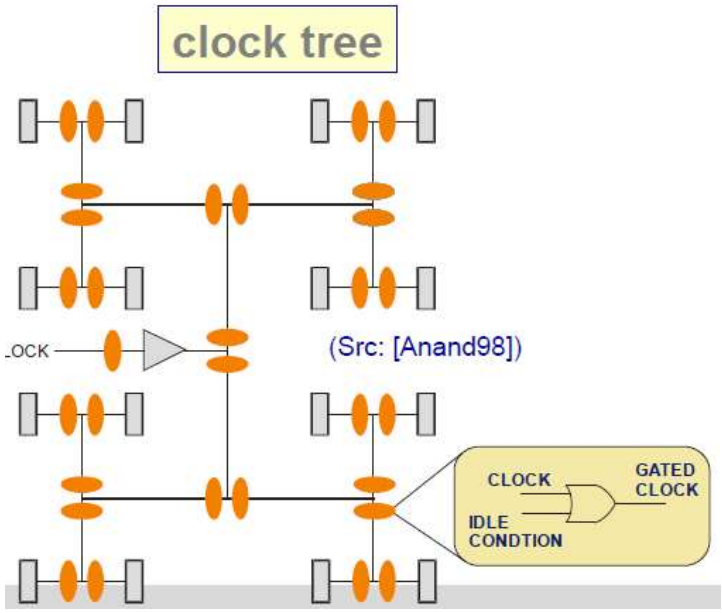
- Often in data paths: output of register is fed back as one of the data inputs through, for example, a multiplexer network
- Task: find condition under which this is the case by traversing the path through the multiplexer network.
- Note: select signals are already present in network => only invert (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.




Source: Raghunathan [1998]

# Clock Gating 7 (For Data Paths)

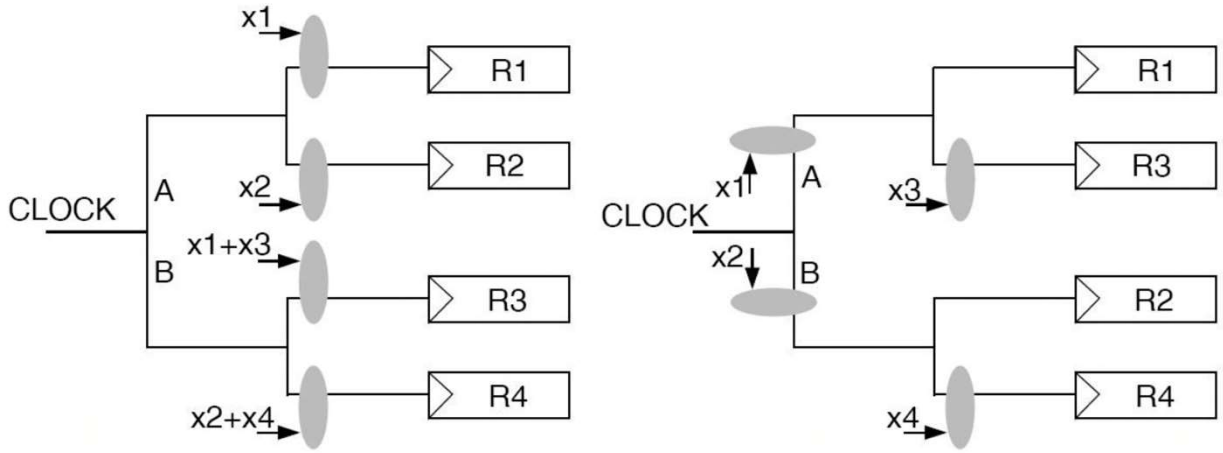
- Shown: a clock tree that has gated clock conditions at various points (levels).
- Tradeoff:
  - Disabling clock at higher level in the tree => a larger capacitance (sum of all smaller ones) is prevented from switching.
  - But: clock transition at certain level can only be suppressed if all registers 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 …).




Source: Raghunathan [1998]

# Clock Gating 8 (Clock Tree Construction)

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




Source: Raghunathan [1998]

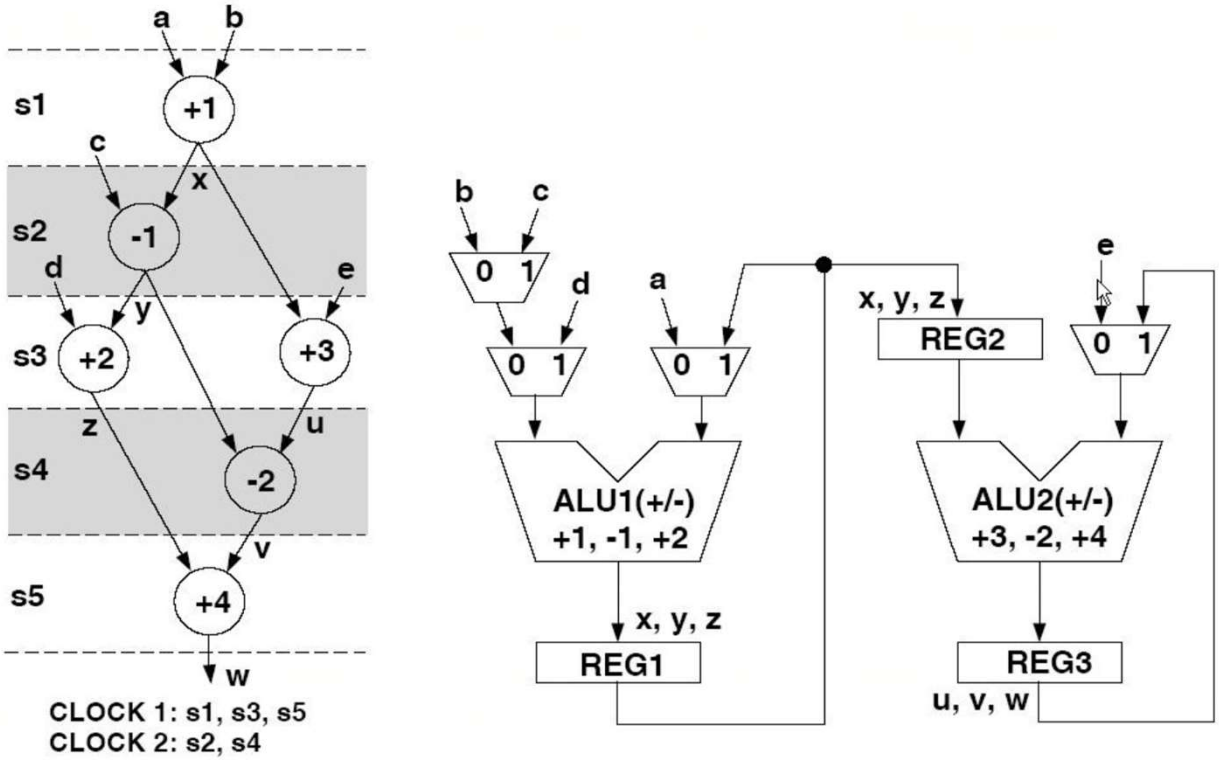
# Clock Gating 9 (Multiple Clocks)

- 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: $C_1 + C_2 < 2C$.
  - 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.

Source: Raghunathan [1998]

# Clock Gating 10 (Multiple Clocks)

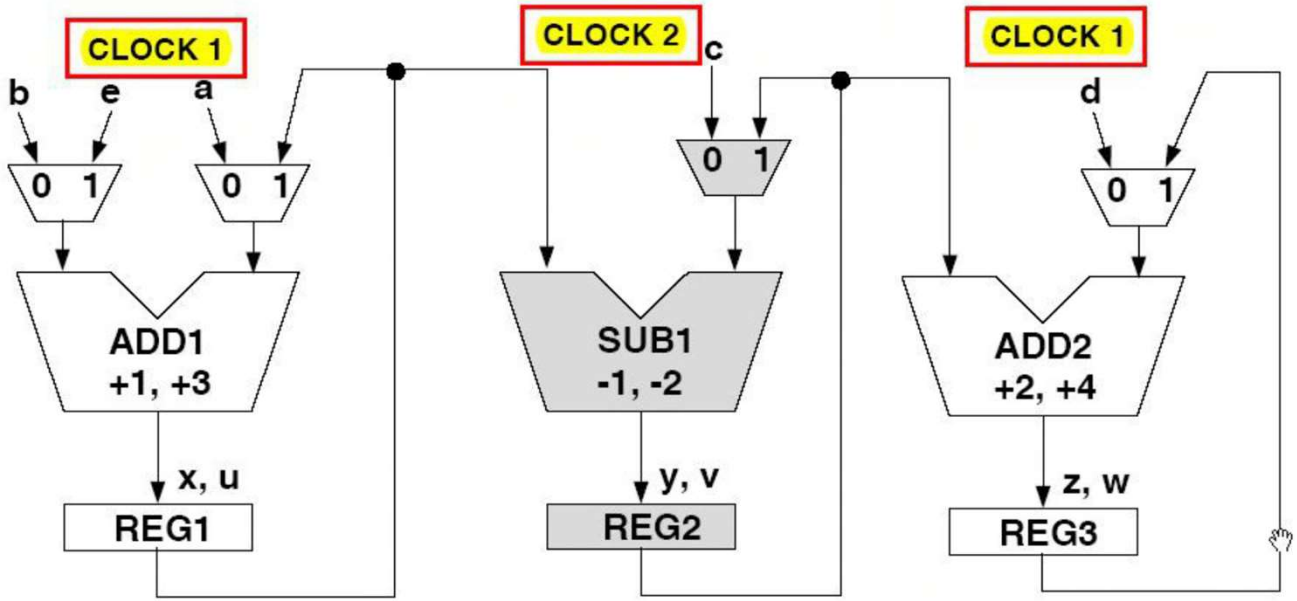
- Idea: using multiple non-overlapping clocks. Example:
  - Scheduled DFG (left fig.): clock cycles of the schedule s1,…,s5 have been assigned to two non-overlapping lock domains, CLOCK1, CLOCK2 in alternating way.
  - Right fig: shows single-clock RTL circuit that implements given DFG using minimal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




Source: Raghunathan [1998]

# Clock Gating 11 (Multiple Clocks)

- Shown: RTL circuit that has been implemented with two clocks.
- Restrictions:
  - an op scheduled in CLOCK1 cannot share an FU with an op in CLOCK2.
  - a variable generated in CLOCK1 cannot share an FU with a variable generated in CLOCK2.
- Why?
  - Ensure that each register can be clocked by either CLOCK1 or CLOCK2.
  - Data path can be partitioned into two domains such that there is only switching activity in their respective active clock cycles.




Source: Raghunathan [1998]

# Clock Gating 12 (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.

# Source

- Raghunathan, Anand, Niraj K. Jha, and Sujit Dey. *High-level power analysis and optimization*. Springer Science & Business Media, 2012.
- Homework >> San Martin, Raul, and John P. Knight. "Power-profiler: optimizing ASICs power consumption at the behavioral level." Proceedings of the 32nd annual ACM/IEEE Design Automation Conference. ACM, 1995.
- Mehra, Renu, and Jan Rabaey. "Exploiting regularity for low-power design." Proceedings of the 1996 IEEE/ACM international conference on Computer-aided design. IEEE Computer Society, 1997.
- Raghunathan, Anand, Sujit Dey, and Niraj K. Jha. "Register transfer level power optimization with emphasis on glitch analysis and reduction." *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 18.8 (1999): 1114-1131.